亚马逊云科技

User Guide

Amazon CloudWatch

Amazon CloudWatch: User Guide

Table of Contents

What is Amazon CloudWatch?

Amazon CloudWatch monitors your Amazon Web Services (Amazon) resources and the applications you run on Amazon in real time, and offers many tools to give you system-wide observability of your application performance, operational health, and resource utilization.

Topics

Operational visibility with metrics, alarms, and dashboards

Metrics collect and track key performance data at user-defined intervals. Many Amazon services automatically report metrics into CloudWatch, and you can also publish custom metrics in CloudWatch from your applications.

Dashboards offer a unified view of your resources and applications with visualizations of your metrics and logs in a single location. You can also share dashboards across accounts and Regions for enhanced operational awareness. CloudWatch provides curated automatic dashboards for many Amazon services, so that you don't have to build them yourself.

You can set up alarms that continuously monitor CloudWatch metrics against user-defined thresholds. They can automatically alert you to breaches of the thresholds, and can also automatically respond to changes in your resources' behavior by triggering automated actions.

Application performance monitoring (APM)

With Application Signals you can automatically detect and monitor your applications' key performance indicators like latency, error rates, and request rates without manual instrumentation or code changes. Application Signals also provides curated dashboards so you can begin monitoring with a minimum of setup.

CloudWatch Synthetics complements this by enabling you to proactively monitor your endpoints and APIs through configurable scripts called *canaries* that simulate user behavior and alert you to availability issues or performance degradation before they impact real users. You can also use CloudWatch RUM to gather performance data from real user sessions.

Use Service Level Objectives (SLOs) in CloudWatch to define, track, and alert on specific reliability targets for your applications, helping you maintain service quality commitments by setting error budgets and monitoring SLO compliance over time.

Infrastructure monitoring

Many Amazon services automatically send basic metrics to CloudWatch for free. Services that send metrics are listed here. Additionally, CloudWatch provides additional monitoring capabilities for several key pieces of Amazon infrastructure:

- Database Insights allows you to monitor database performance metrics in real time, analyze SQL query performance, and troubleshoot database load issues for Amazon database services.
- Lambda Insights provides system-level metrics for Lambda functions, including memory and CPU utilization tracking, and cold start detection and analysis.
- Container Insights allows you to collect and analyze metrics from containerized applications, on Amazon ECS clusters, Amazon EKS clusters, and self-managed Kubernetes clusters on Amazon EC2.

Collect, store, and query logs

CloudWatch Logs offers a suite of powerful features for comprehensive log management and analysis. Logs ingested from Amazon services and custom applications are stored in log groups and streams for easy organization. Use CloudWatch Logs Insights to perform interactive, fast queries on your log data, with a choice of three query languages including SQL and PPL. Use log anomaly detection to find unusual patterns in log events in a log group, which can indicate issues. Create metric filters to extract numerical values from logs and generate CloudWatch metrics, which you can use for alerting and dashboards. Set up subscription filters to process and analyze logs in real-time or route them to other services like Amazon S3 or Firehose.

Use the CloudWatch agent to gather metrics, logs, and traces from Amazon EC2 fleets

Use the CloudWatch agent to collect detailed system metrics about processes, CPU, memory, disk usage, and network performance from your fleets of Amazon EC2 instances and on-premises servers. You can also collect and monitor custom metrics from your applications, aggregate logs from multiple sources, and configure alarms based on the collected data. You can also use the agent to gather GPU metrics. The agent supports both Windows and Linux operating systems and can integrate with Systems Manager for centralized configuration management.

Cross-account monitoring

CloudWatch cross-account observability lets you set up a central monitoring account to monitor and troubleshoot applications that span multiple accounts. From the central account, you can view metrics, logs, and traces from source accounts across your organization. This centralized approach enables you to create cross-account dashboards, set up alarms that watch metrics from multiple accounts, and perform root-cause analysis across account boundaries. With CloudWatch cross-account observability, you can link source accounts either individually or link them automatically through Amazon Organizations.

Solutions catalog

CloudWatch offers a catalog of readily available configurations to help you quickly implement monitoring for various Amazon services and common workloads, such as Java Virtual Machines (JVM), NVIDIA GPU, Apache Kafka, Apache Tomcat, and NGINX. These solutions provide focused guidance, including instructions for installing and configuring the CloudWatch agent, deploying pre-defined custom dashboards, and setting up related alarms.

Network and internet monitoring

CloudWatch provides comprehensive network and internet monitoring capabilities through CloudWatch Network Monitoring.

Internet Monitor uses Amazon global networking data to analyze internet performance and availability between your applications and end users. With an internet monitor, you can identify or get notifications for increased latency or regional disruptions that impact your customers. Internet monitors work by analyzing your VPC flow logs to provide automated insights about network traffic patterns and performance. You can also get suggestions for how to optimize application performance for your clients.

Network Flow Monitor displays network performance information gathered by lightweight software agents that you install on your instances. Using a flow monitor, you can quickly visualize packet loss and latency of your network connections over a time frame that you specify. Each monitor also generates a network health indicator (NHI), which tells you whether there were Amazon network issues for the network flows tracked by your monitor during the time period that you're evaluating.

When you connect by using Amazon Direct Connect, you can use synthetic monitors in Network Synthetic Monitor to proactively monitor network connectivity by running synthetic tests between a VPC and on-premises endpoints. When you create a synthetic monitor, you specify probes by providing a VPC subnet and on-premises IP addresses. Amazon creates and manages the infrastructure in the background that is required to perform round-trip time and packet loss measurements with the probes. These tests detect issues with connectivity, DNS, and latency before they impact your applications, so that you can take action to improve your end users' experience.

Billing and costs

For complete information about CloudWatch pricing, see Amazon CloudWatch Pricing.

For information that can help you analyze your bill and possibly optimize and reduce costs, see Analyzing, optimizing, and reducing CloudWatch costs.

Amazon CloudWatch resources

The following related resources can help you as you work with this service.

Resource	Description
Amazon CloudWatch FAQs	The FAQ covers the top questions developers have asked about this product.

Resource	Description
Amazon Developer Center	A central starting point to find documentation, code examples, release notes, and other information to help you build innovative applications with Amazon.
Amazon Web Services Management Console	The console allows you to perform most of the functions of Amazon CloudWatch and various other Amazon offerings without programming.
Amazon CloudWatch Discussion Forums	Community-based forum for developers to discuss technical questions related to Amazon CloudWatch.
Amazon Web Services Support	The hub for creating and managing your Amazon Web Services Support cases. Also includes links to other helpful resources, such as forums, technical FAQs, service health status, and Amazon Trusted Advisor.
Amazon CloudWatch product i nformation	The primary web page for information about Amazon CloudWatch.
Contact Us	A central contact point for inquiries concerning Amazon billing, account, events, abuse, etc.

Getting set up

To use Amazon CloudWatch you need an Amazon account. Your Amazon account allows you to use services (for example, Amazon EC2) to generate metrics that you can view in the CloudWatch console, a point-and-click web-based interface. In addition, you can install and configure the Amazon command line interface (CLI).

Sign up for an Amazon Web Services account

If you do not have an Amazon Web Services account, use the following procedure to create one.

To sign up for Amazon Web Services

1. Open http://www.amazonaws.cn/ and choose **Sign Up**.
2. Follow the on-screen instructions.

Amazon sends you a confirmation email after the sign-up process is complete. At any time, you can view your current account activity and manage your account by going to http://www.amazonaws.cn/ and choosing **My Account**.

Secure IAM users

After you sign up for an Amazon Web Services account, safeguard your administrative user by turning on multi-factor authentication (MFA). For instructions, see Enable a virtual MFA device for an IAM user (console) in the *IAM User Guide*.

To give other users access to your Amazon Web Services account resources, create IAM users. To secure your IAM users, turn on MFA and only give the IAM users the permissions needed to perform their tasks.

For more information about creating and securing IAM users, see the following topics in the *IAM User Guide*:

- Creating an IAM user in your Amazon Web Services account
- Access management for Amazon resources
- Example IAM identity-based policies

Sign in to the Amazon CloudWatch console

To sign in to the Amazon CloudWatch console

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. If necessary, use the navigation bar to change the Region to the Region where you have your Amazon resources.
3. Even if this is the first time you are using the CloudWatch console, **Your Metrics** could already report metrics, because you have used an Amazon product that automatically pushes metrics to Amazon CloudWatch for free. Other services require that you enable metrics.

   If you do not have any alarms, the **Your Alarms** section will have a **Create Alarm** button.

Set up the Amazon CLI

You can use the Amazon CLI or the Amazon CloudWatch CLI to perform CloudWatch commands. Note that the Amazon CLI replaces the CloudWatch CLI; we include new CloudWatch features only in the Amazon CLI.

For information about how to install and configure the Amazon CLI, see Getting Set Up with the Amazon Command Line Interface in the *Amazon Command Line Interface User Guide*.

For information about how to install and configure the Amazon CloudWatch CLI, see Set Up the Command Line Interface in the *Amazon CloudWatch CLI Reference*.

Analyzing, optimizing, and reducing CloudWatch costs

This section describes how Amazon CloudWatch features generate costs. It also provides methods that can help you analyze, optimize, and reduce CloudWatch costs. Throughout this section, we sometimes refer to pricing when describing CloudWatch features. For information about pricing, see Amazon CloudWatch pricing.

Topics

Analyze CloudWatch cost and usage data with Cost Explorer

With Amazon Cost Explorer, you can visualize and analyze cost and usage data for Amazon Web Services services over time, including CloudWatch. For more information, see Getting started with Amazon Cost Explorer.

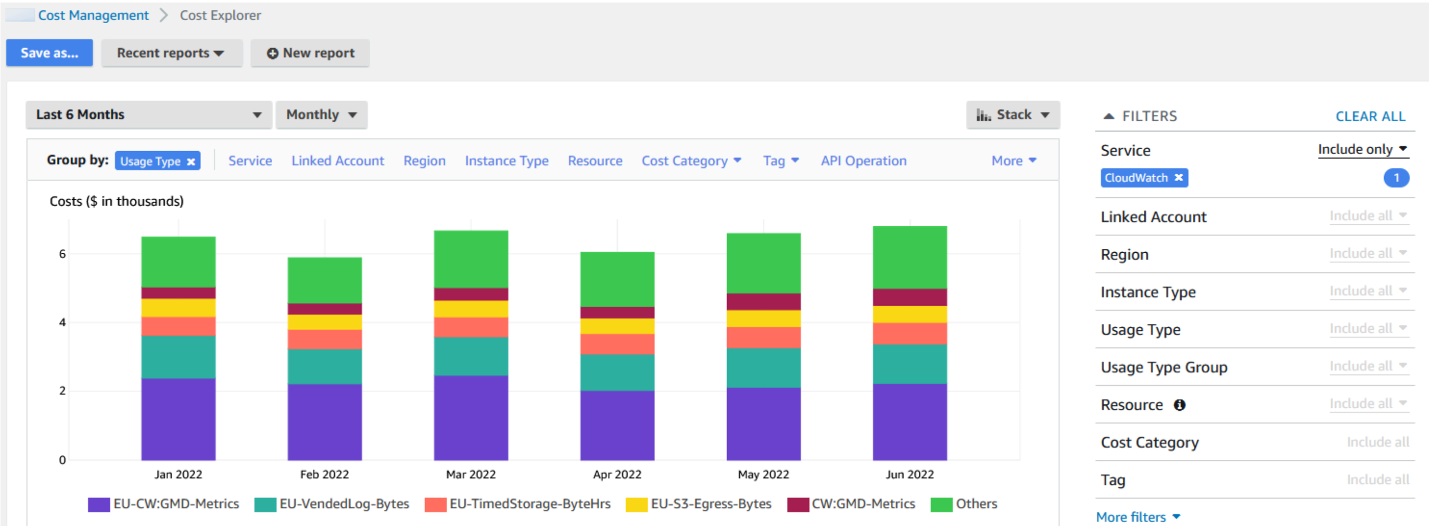
The following procedure describes how to use Cost Explorer to visualize and analyze CloudWatch cost and usage data.

To visualize and analyze CloudWatch cost and usage data

1. Sign in to the Cost Explorer console at https://console.aws.amazon.com/cost-management/home#/custom.
2. Under **FILTERS**, for **Service**, select **CloudWatch**.
3. For **Group by**, choose **Usage Type**. You can also group your results by other categories, such as the following:

   - **API Operation** – See which API operations generated the most costs.
   - **Region** – See which Regions generated the most costs.

The following image shows an example of the costs that CloudWatch features generated over six months.



To see which CloudWatch features generated the most costs, look at the values for `UsageType`. For example, `EU-CW:GMD-Metrics` represents the costs that CloudWatch bulk API requests generated.

> **Note**
>
> The strings for `UsageType` match specific features and Regions. For example, the first part of `EU-CW:GMD-Metrics` (EU) matches the Europe (Ireland) Region, and the second part of `EU-CW:GMD-Metrics` (`GMD-Metrics`) matches CloudWatch bulk API requests.
> The entire string for `UsageType` can be formatted as follows: `<Region>-CW:<Feature>` or `<Region>-<Feature>`.
> Some CloudWatch features such as logs and alarms also uses the `Global` Region to identify the Free Tier usage. For example, `Global-DataScanned-Bytes` represents free CloudWatch Logs data ingestion usage.
> To enhance readability, the strings for `UsageType` in the tables throughout this document have been shortened to their string suffixes. For example, `EU-CW:GMD-Metrics` is shortened to `GMD-Metrics`.

The following table includes the names of each CloudWatch feature, lists the names of each sub-feature, and lists the strings for `UsageType`.

CloudWatch feature	*CloudWatch sub-feature*	`UsageType`
CloudWatch metrics	*Custom metrics*	`MetricMonitorUsage`
	Detailed monitoring	`MetricMonitorUsage`
	Embedded metrics	`MetricMonitorUsage`
CloudWatch API requests	*API requests*	`Requests`
	Bulk (Get)	`GMD-Metrics`
	Contributor Insights	`GIRR-Metrics`
	Bitmap image snapshot	`GMWI-Metrics`
CloudWatch metric streams	*Metric streams*	`MetricStreamUsage`
CloudWatch dashboards	*Dashboard with 50 or fewer metrics*	`DashboardsUsageHour-Basic`
	Dashboard with more than 50 metrics	`DashboardsUsageHour`
CloudWatch alarms	*Standard (metric alarm)*	`AlarmMonitorUsage`
	High resolution (metric alarm)	`HighResAlarmMonitorUsage`
	Metrics Insights query alarm	`MetricInsightAlarmUsage`

CloudWatch feature	***CloudWatch sub-feature***	**UsageType**
	Composite (aggregated alarm)	`CompositeAlarmMonitorUsage`
Container Insights	Enhanced observability for Amazon EKS	`ObservationUsage`
	Enhanced observability for Amazon ECS	`MetricsUsage`
CloudWatch Application Signals	*Application Signals*	`Application-Signals`
CloudWatch custom logs	*Collect (data ingestion for Standard log class)*	`DataProcessing-Bytes`
	Collect (data ingestion for Infrequent Access log class)	`DataProcessingIA-Bytes`
	Analyze (query)	`DataScanned-Bytes`
	Analyze (Live Tail)	`Logs-LiveTail`
	Store (archive)	`TimedStorage-ByteHrs`
	Detect and mask (data protection)	`DataProtection-Bytes`
CloudWatch vended logs	*Delivery (Amazon CloudWatch Logs Standard log class)*	`VendedLog-Bytes`
	Delivery (CloudWatch Logs Infrequent Access log class)	`VendedLogIA-Bytes`

CloudWatch feature	*CloudWatch sub-feature*	UsageType
	Delivery (Amazon S3)	`S3-Egress-Bytes`
	Delivery (Amazon S3) in Parquet format	`S3-Egress-InputBytes`
	Delivery (Amazon Data Firehose)	`FH-Egress-Bytes`
Contributor Insights	*CloudWatch Logs (Rules)*	`ContributorInsight Rules`
	CloudWatch Logs (Events)	`ContributorInsight Events`
	Amazon DynamoDB (Rules)	`ContributorRulesMa naged`
	DynamoDB Events)	`ContributorEventsM anaged`
Canaries (Synthetics)	*Run*	`Canary-runs`
Evidently	*Events*	`Evidently-event`
	Analysis Units	`Evidently-eau`
RUM	*Events*	`RUM-event`

CloudWatch feature	CloudWatch sub-feature	UsageType
Network monitoring	*Network Synthetic Monitor*	`CWNMHybrid-Paid`
	Internet Monitor (monitored resources)	`InternetMonitor-MonitoredResource`
	Internet Monitor (monitored city networks)	`InternetMonitor-CityNetwork`

Analyze CloudWatch cost and usage data with Amazon Cost and Usage Reports and Athena

Another way to analyze CloudWatch cost and usage data is by using Amazon Cost and Usage Reports with Amazon Athena. Amazon Cost and Usage Reports contain a comprehensive set of cost and usage data. You can create reports that track your costs and usage, and you can publish these reports to an S3 bucket of your choice. You also can download and delete your reports from your S3 bucket. For more information, see What are Amazon Cost and Usage Reports? in the *Amazon Cost and Usage Reports User Guide*.

> **Note**
>
> There is no charge for using Amazon Cost and Usage Reports. You only pay for storage when you publish your reports to Amazon Simple Storage Service (Amazon S3). For more information, see Quotas and restrictions in the *Amazon Cost and Usage Reports User Guide*.

Athena is a query service that you can use with Amazon Cost and Usage Reports to analyze cost and usage data. You can query your reports in your S3 bucket without needing to download them first. For more information, see What is Amazon Athena? in the *Amazon Athena User Guide*. For more information, see What is Amazon Athena? in the *Amazon Athena User Guide*. For information about pricing, see Amazon Athena pricing.

The following procedure describes the process for enabling Amazon Cost and Usage Reports and integrating the service with Athena. The procedure contains two example queries that you can use to analyze CloudWatch cost and usage data.

> **Note**
>
> You can use any of the example queries in this document. All of the example queries in this document correspond to a database named ***costandusagereport***, and show results for the month of April and the year 2022. You can change this information. However, before you run a query, make sure that the name of your database matches the name of the database in the query.

To analyze cost and usage data with Amazon Cost and Usage Reports and Athena

1. Enable Amazon Cost and Usage Reports. For more information, see Creating cost and usage reports in the *Amazon Cost and Usage Reports User Guide*.

   > **Tip**
   >
   > When you create your reports, make sure to select **Include resource IDs**. Otherwise, your reports won't include the column `line_item_resource_id`. This line helps you further identify costs when analyzing cost and usage data.

2. Integrate Amazon Cost and Usage Reports with Athena. For more information, see Setting up Athena using Amazon CloudFormation templates in the *Amazon Cost and Usage Reports User Guide*.
3. Query your cost and usage reports.

Example of Athena query to show CloudWatch costs per month

You can use the following query to show which CloudWatch features generated the most costs for a given month.

```
SELECT
CASE
-- Metrics
```

```
WHEN line_item_usage_type LIKE '%%MetricMonitorUsage%%' THEN 'Metrics (Custom, Detailed
 monitoring management portal EMF)'
WHEN line_item_usage_type LIKE '%%Requests%%' THEN 'Metrics (API Requests)'
WHEN line_item_usage_type LIKE '%%GMD-Metrics%%' THEN 'Metrics (Bulk API Requests)'
WHEN line_item_usage_type LIKE '%%MetricStreamUsage%%' THEN 'Metric Streams'
-- Contributor Insights
WHEN line_item_usage_type LIKE '%%Contributor%%' THEN 'Contributor Insights'
-- Dashboard
WHEN line_item_usage_type LIKE '%%DashboardsUsageHour%%' THEN 'Dashboards'
-- Alarms
WHEN line_item_usage_type LIKE '%%AlarmMonitorUsage%%' THEN 'Alarms (Standard)'
WHEN line_item_usage_type LIKE '%%HighResAlarmMonitorUsage%%' THEN 'Alarms (High
 Resolution)'
WHEN line_item_usage_type LIKE '%%MetricInsightAlarmUsage%%' THEN 'Alarms (Metrics
 Insights)'
WHEN line_item_usage_type LIKE '%%CompositeAlarmMonitorUsage%%' THEN 'Alarms
 (Composite)'
-- Container Insights with enhanced observability
WHEN (line_item_usage_type LIKE '%%MetricsUsage%%' OR line_item_usage_type LIKE '%
%ObservationUsage%%') THEN 'Container Insights (Enhanced Observability)'
-- Logs
WHEN line_item_usage_type LIKE '%%DataProcessing-Bytes%%' THEN 'Logs (Collect - Data
 Ingestion)'
WHEN line_item_usage_type LIKE '%%DataProcessingIA-Bytes%%' THEN 'Infrequent Access
 Logs (Collect - Data Ingestion)'
WHEN line_item_usage_type LIKE '%%DataProtection-Bytes%%' THEN 'Logs (Data Protection -
 Detect and Mask)'
WHEN line_item_usage_type LIKE '%%TimedStorage-ByteHrs%%' THEN 'Logs (Storage -
 Archival)'
WHEN line_item_usage_type LIKE '%%DataScanned-Bytes%%' THEN 'Logs (Analyze - Logs
 Insights queries)'
WHEN line_item_usage_type LIKE '%%Logs-LiveTail%%' THEN 'Logs (Analyze - Logs Live
 Tail)'
-- Vended Logs
WHEN line_item_usage_type LIKE '%%VendedLog-Bytes%%' THEN 'Vended Logs (Delivered to
 CW)'
WHEN line_item_usage_type LIKE '%%VendedLogIA-Bytes%%' THEN 'Vended Infrequent Access
 Logs (Delivered to CW)'
WHEN line_item_usage_type LIKE '%%FH-Egress-Bytes%%' THEN 'Vended Logs (Delivered to
 Data Firehose)'
WHEN (line_item_usage_type LIKE '%%S3-Egress%%') THEN 'Vended Logs (Delivered to S3)'
-- Network Monitoring
WHEN line_item_usage_type LIKE '%%CWNMHybrid-Paid%%' THEN 'Network Monitor'
WHEN line_item_usage_type LIKE '%%InternetMonitor%%' THEN 'Internet Monitor'
```

```
-- Other
WHEN line_item_usage_type LIKE '%%Application-Signals%%' THEN 'Application Signals'
WHEN line_item_usage_type LIKE '%%Canary-runs%%' THEN 'Synthetics'
WHEN line_item_usage_type LIKE '%%Evidently%%' THEN 'Evidently'
WHEN line_item_usage_type LIKE '%%RUM-event%%' THEN 'RUM'
ELSE 'Others'
END AS UsageType,
-- REGEXP_EXTRACT(line_item_resource_id,'^(?:.+?:){5}(.+)$',1) as ResourceID,
SUM(CAST(line_item_usage_amount AS double)) AS UsageQuantity,
 SUM(CAST(line_item_unblended_cost AS decimal(16,8))) AS TotalSpend
FROM
costandusagereport
WHERE product_product_name = 'AmazonCloudWatch'
AND year='2022'
AND month='4'
AND line_item_line_item_type NOT IN
 ('Tax','Credit','Refund','EdpDiscount','Fee','RIFee')
-- AND line_item_usage_account_id = '123456789012' – If you want to filter on a
 specific account, you can remove this comment at the beginning of the line and specify
 an AWS account.
GROUP BY
1
ORDER BY TotalSpend DESC,
UsageType;
```

Example of Athena query to show how CloudWatch features generated costs

You can use the following query to show the results for `UsageType` and `Operation`. This shows you how CloudWatch features generated costs. The results also show the values for `UsageQuantity` and `TotalSpend`, so that you can see your total usage costs.

> **Tip**
>
> For more information about `UsageType`, add the following line to this query:
> `line_item_line_item_description`
> This line creates a column called ***Description***.

```
SELECT
bill_payer_account_id as Payer,
line_item_usage_account_id as LinkedAccount,
```

```
line_item_usage_type AS UsageType,
line_item_operation AS Operation,
line_item_resource_id AS ResourceID,
SUM(CAST(line_item_usage_amount AS double)) AS UsageQuantity,
SUM(CAST(line_item_unblended_cost AS decimal(16,8))) AS TotalSpend
FROM
costandusagereport
WHERE
product_product_name = 'AmazonCloudWatch'
AND year='2022'
AND month='4'
AND line_item_line_item_type NOT IN
 ('Tax','Credit','Refund','EdpDiscount','Fee','RIFee')
GROUP BY
bill_payer_account_id,
line_item_usage_account_id,
line_item_usage_type,
line_item_resource_id,
line_item_operation
```

Optimizing and reducing costs of CloudWatch metrics

Many Amazon Web Services services, such as Amazon Elastic Compute Cloud (Amazon EC2) , Amazon S3, and Amazon Data Firehose, automatically send metrics to CloudWatch at no charge. However, metrics that are grouped in the following categories can incur additional costs:

- ***Custom metrics, detailed monitoring, and embedded metrics***
- ***API requests***
- ***Metric streams***

For more information, see Using Amazon CloudWatch metrics.

Custom metrics

You can create custom metrics to organize data points in any order and at any rate.

All custom metrics are prorated by the hour. They're metered only when they're sent to CloudWatch. For information about how metrics are priced, see Amazon CloudWatch Pricing.

The following table lists the names of relevant sub-features for CloudWatch metrics. The table includes the strings for `UsageType` and `Operation`, which can help you analyze and identify metric-related costs.

> **Note**
>
> To get more details about the metrics that are listed in the following table while you're querying cost and usage data with Athena, match the strings for `Operation` with the results that are shown for `line_item_operation`.

CloudWatch sub-feature	**`UsageType`**	**`Operation`**	**Purpose**
Custom metrics	`MetricMonitorUsage`	`MetricStorage`	Custom metrics
Detailed monitoring	`MetricMonitorUsage`	`MetricStorage:AWS/` *`{Service}`*	Detailed monitoring
Embedded metrics	`MetricMonitorUsage`	`MetricStorage:Amazon/Logs-EMF`	Logs embedded metrics
Log filters	`MetricMonitorUsage`	`MetricStorage:Amazon/CloudWatchLogs`	Log group metric filters

Detailed monitoring

CloudWatch has two types of monitoring:

- ***Basic monitoring***

  Basic monitoring is free and automatically enabled for all Amazon Web Services services that support the feature.

- ***Detailed monitoring***

  Detailed monitoring incurs costs and adds different enhancements depending on the Amazon Web Services service. For each Amazon Web Services service that supports detailed monitoring, you can choose whether to enable it for that service. For more information, see Basic and detailed monitoring.

> **Note**
>
> Other Amazon Web Services services support detailed monitoring and might refer to this feature using a different name. For example, detailed monitoring for Amazon S3 is referred to as ***request metrics***.

Similar to custom metrics, detailed monitoring is prorated by the hour and metered only when data is sent to CloudWatch. Detailed monitoring generates costs by the number of metrics that are sent to CloudWatch. To reduce costs, only enable detailed monitoring when necessary. For information about how detailed monitoring is priced, see Amazon CloudWatch Pricing.

Example: Athena query

You can use the following query to show which EC2 instances have detailed monitoring enabled.

```
SELECT
bill_payer_account_id as Payer,
line_item_usage_account_id as LinkedAccount,
line_item_usage_type AS UsageType,
line_item_operation AS Operation,
line_item_resource_id AS ResourceID,
SUM(CAST(line_item_usage_amount AS double)) AS UsageQuantity,
SUM(CAST(line_item_unblended_cost AS decimal(16,8))) AS TotalSpend
FROM
costandusagereport
WHERE
product_product_name = 'AmazonCloudWatch'
AND year='2022'
AND month='4'
AND line_item_operation='MetricStorage:AWS/EC2'
AND line_item_line_item_type NOT IN
 ('Tax','Credit','Refund','EdpDiscount','Fee','RIFee')
```

```
GROUP BY
bill_payer_account_id,
line_item_usage_account_id,
line_item_usage_type,
line_item_resource_id,
line_item_operation,
line_item_line_item_description
ORDER BY line_item_operation
```

Embedded metrics

With the CloudWatch embedded metric format, you can ingest application data as log data, so that you can generate actionable metrics. For more information, see Ingesting high-cardinality logs and generating metrics with the CloudWatch embedded metric format.

Embedded metrics generate costs by the number of logs ingested, number of logs archived, and number of custom metrics generated.

The following table lists the names of relevant sub-features for the CloudWatch embedded metric format. The table includes the strings for `UsageType` and `Operation`, which can help you analyze and identify costs.

CloudWatch subfeature	**`UsageType`**	**`Operation`**	**Purpose**
Custom metrics	`MetricMonitorUsage`	`MetricStorage:AWS/Logs-EMF`	Logs embedded metrics
Logs ingestion	`DataProcessing-Bytes`	`PutLogEvents`	Uploads a batch of log events to the specified log group or log stream
Logs archival	`TimedStorage-ByteHrs`	`HourlyStorageMetering`	Stores logs per hour and logs per byte in CloudWatch Logs

To analyze costs, use Amazon Cost and Usage Reports with Athena so that you can identify which metrics are generating costs and determine how the costs are generated.

To make the most of costs generated by the CloudWatch embedded metric format, avoid creating metrics based on high-cardinality dimensions. This way, CloudWatch doesn't create a custom metric for each unique dimension combination. For more information, see Dimensions.

API requests

CloudWatch has the following types of API requests:

- ***API requests***
- ***Bulk (Get)***
- ***Contributor Insights***
- ***Bitmap image snapshot***

API requests generate costs by the request type and number of metrics requested.

The following table lists the types of API requests and includes the strings for `UsageType` and `Operation`, which can help you analyze and identify API-related costs.

API request type	**`UsageType`**	**`Operation`**	**Purpose**
API requests	`Requests`	`GetMetricStatistics`	Retrieves statistics for the specified metrics
	`Requests`	`ListMetrics`	Lists the specified metrics
	`Requests`	`PutMetricData`	Publishes metric data points to CloudWatch
	`Requests`	`GetDashboard`	Displays details for the specified dashboards

API request type	`UsageType`	`Operation`	Purpose
	`Requests`	`ListDashboards`	Lists the dashboards in your account
	`Requests`	`PutDashboard`	Creates or updates a dashboard
	`Requests`	`DeleteDashboards`	Deletes all specified dashboards
Bulk (Get)	`GMD-Metrics`	`GetMetricData`	Retrieves CloudWatch metric values
Contributor Insights	`GIRR-Metrics`	`GetInsightRuleReport`	Returns time-series data that's collected by a Contributor Insights rule
Bitmap image snapshot	`GMWI-Metrics`	`GetMetricWidgetImage`	Retrieves a snapshot of one or more CloudWatch metrics as a bitmap image

To analyze costs, use Cost Explorer, and group your results by ***API Operation***.

Costs for API requests vary, and you incur costs when you exceed the number of API calls provided to you under the Amazon Free Tier limit.

> **Note**
>
> `GetMetricData` and `GetMetricWidgetImage` aren't included under the Amazon Free Tier limit. For more information, see Using the Amazon Free Tier in the *Amazon Billing User Guide*.

The API requests that typically drive cost are `Put` and `Get` requests.

`PutMetricData`

`PutMetricData` generates costs every time that it's called and can incur significant costs depending on the use case. For more information, see [PutMetricData] in the *Amazon CloudWatch API Reference*.

To make the most of costs that are generated by `PutMetricData`, batch more data into your API calls. Depending on your use case, consider using CloudWatch Logs or the CloudWatch embedded metric format to inject metric data. For more information, see the following resources:

- What is Amazon CloudWatch Logs? in the *Amazon CloudWatch Logs User Guide*
- Ingesting high-cardinality logs and generating metrics with CloudWatch embedded metric format
- Lowering costs and focusing on our customers with Amazon CloudWatch embedded custom metrics

`GetMetricData`

`GetMetricData` can also generate significant costs. Common use cases that drive cost involve third-party monitoring tools that pull data to generate insights. For more information, see GetMetricData in the *Amazon CloudWatch API Reference*.

To reduce costs generated by `GetMetricData`, consider only pulling data that's monitored and used, or consider pulling data less often. Depending on your use case, you might consider using metric streams instead of `GetMetricData`, so that you can push data in near real time to third parties at a lower cost. For more information, see the following resources:

- Using metric streams
- CloudWatch Metric Streams - Send Amazon Metrics to Partners and to Your Apps in Real Time

`GetMetricStatistics`

Depending on your use case, you might consider using `GetMetricStatistics` instead of `GetMetricData`. With `GetMetricData`, you can retrieve data quickly and at scale. However, `GetMetricStatistics` is included under the Amazon Free Tier limit for up to one million API requests, which can help you reduce costs if you don't need to retrieve as many metrics and data points per call. For more information, see the following resources:

- GetMetricStatistics in the *Amazon CloudWatch API Reference*
- Should I use GetMetricData or GetMetricStatistics?

> **Note**
>
> External callers make API calls. For APIs that are supported by CloudTrail data events (such as **GetMetricData** and **GetMetricWidgetImage**), you can use CloudTrail to identify the top CloudWatch API callers and potentially mitigate or identify unexpected calls. For more information, see How to use CloudTrail to analyse your CloudWatch API Usage. For other CloudWatch APIs not supported by CloudTrail, you can open a technical support request to the CloudWatch team and ask for information about them. For information about creating a technical support request, see How do I get technical support from Amazon?.

CloudWatch metric streams

With CloudWatch metric streams, you can send metrics continuously to Amazon destinations and third-party service provider destinations.

Metric streams generate costs by the number of metric updates. Metric updates always include values for the following statistics:

- `Minimum`
- `Maximum`
- `Sample Count`
- `Sum`

For more information, see Statistics that can be streamed.

To analyze costs that are generated by CloudWatch metric streams, use Amazon Cost and Usage Reports with Athena. This way, you can identify which metric streams are generating costs and determine how the costs are generated.

Example: Athena query

You can use the following query to track which metric streams generate costs by Amazon Resource Name (ARN).

```
SELECT
SPLIT_PART(line_item_resource_id,'/',2) AS "Stream Name",
line_item_resource_id as ARN,
```

```
SUM(CAST(line_item_unblended_cost AS decimal(16,2))) AS TotalSpend
FROM
costandusagereport
WHERE
product_product_name = 'AmazonCloudWatch'
AND year='2022'
AND month='4'
AND line_item_line_item_type NOT IN
 ('Tax','Credit','Refund','EdpDiscount','Fee','RIFee')
-- AND line_item_usage_account_id = '123456789012' – If you want to filter on a
 specific account, you can
remove this comment at the beginning of the line and specify an AWS account.
AND line_item_usage_type LIKE '%%MetricStreamUsage%%'
GROUP BY line_item_resource_id
ORDER BY TotalSpend DESC
```

To reduce costs generated by CloudWatch metric streams, stream only the metrics that bring your business value. You also can stop or pause any metric stream that you're not using.

Optimizing and reducing costs of CloudWatch alarms

With CloudWatch alarms, you can create alarms based on a single metric, alarms based on a Metrics Insights query, and composite alarms which watch other alarms.

> **Note**
>
> Costs for metric and composite alarms are prorated by the hour. You incur costs for your alarms only while your alarms exist. To optimize cost, be sure not to leave behind misconfigured or low-value alarms. To help with this, you can automate the cleanup of CloudWatch alarms that you no longer need. For more information, see [Automating Amazon CloudWatch Alarm Cleanup at Scale](https://aws.amazon.com/blogs/mt/automating-amazon-cloudwatch-alarm-cleanup-at-scale/)

Metric alarms

Metric alarms have the following resolution settings:

- **Standard** (evaluated every 60 seconds)
- **High resolution** (evaluated every 10 seconds)

When you create a metric alarm, your costs are based on your alarm’s resolution setting and the number of metrics that your alarm references. For example, a metric alarm that references one metric incurs one alarm-metric cost per hour. For more information, see Using Amazon CloudWatch alarms.

If you create a metric alarm that contains a metric math expression, which references multiple metrics, you incur a cost for each alarm-metric that’s referenced in the metric math expression. For information about how to create a metric alarm that contains a metric math expression, see Creating a CloudWatch alarm based on a metric math expression.

If you create an anomaly detection alarm, where your alarm analyzes past metric data to create a model of expected values, you incur a cost for each alarm-metric that's referenced in your alarm plus two additional metrics, one for the upper and lower band metrics that the anomaly detection model creates. For information about how to create an anomaly detection alarm, see Creating a CloudWatch alarm based on anomaly detection.

Metrics Insights query alarms

Metric Insights query alarms are a specific type of metric alarm, only available with standard resolution (evaluated every 60 seconds).

When you create a Metric Insights query alarm, your costs are based on the number of metrics analyzed by the query that your alarm references. For example, a Metric Insights query alarm that references a query whose filter matches ten metrics incurs ten metrics analyzed cost per hour. For more information, see the pricing example on Amazon CloudWatch Pricing.

If you create an alarm that contains both a Metrics Insights query and a metric math expression, it is reported as a Metrics Insights query alarm. If your alarm contains a metric math expression which references other metrics in addition to the metrics analyzed by the Metrics Insights query, you incur an additional cost for each alarm-metric that’s referenced in the metric math expression. For information about how to create a metric alarm that contains a metric math expression, see Creating a CloudWatch alarm based on a metric math expression.

Composite alarms

Composite alarms contain rule expressions that specify how they should evaluate the states of other alarms to determine their own states. Composite alarms incur a standard cost per hour, regardless of how many other alarms they evaluate. Alarms that composite alarms reference in rule expressions incur separate costs. For more information, see Creating a composite alarm.

Alarm usage types

The following table lists the names of relevant sub-features for CloudWatch alarms. The table includes the strings for `UsageType`, which can help you analyze and identify alarm-related costs.

CloudWatch sub-feature	**`UsageType`**
Standard metric alarm	`AlarmMonitorUsage`
High-resolution metric alarm	`HighResAlarmMonitorUsage`
Metrics Insights query alarm	`MetricInsightAlarmUsage`
Composite alarm	`CompositeAlarmMonitorUsage`

Reducing alarm costs

To optimize costs generated by metric math alarms that aggregate four or more metrics, you can aggregate data before the data is sent to CloudWatch. This way, you can create an alarm for a single metric instead of an alarm that aggregates data for multiple metrics. For more information, see Publishing custom metrics.

To optimize costs generated by Metrics Insights query alarms, you can ensure that the filter used for the query matches only the metrics that you want to monitor.

The best way to reduce costs is to remove all unnecessary or unused alarms. For example, you can delete alarms that evaluate metrics emitted by Amazon resources that no longer exist.

Example of using `DescribeAlarms` to check for alarms in `INSUFFICIENT_DATA` state

If you delete a resource, but not the metric alarms that the resource emits, the alarms still exist and typically will go into the `INSUFFICIENT_DATA` state. To check for alarms that are in the `INSUFFICIENT_DATA` state, use the following Amazon Command Line Interface (Amazon CLI) command.

```
aws cloudwatch describe-alarms –state-value INSUFFICIENT_DATA
```

For more information, see Automating Amazon CloudWatch Alarm Cleanup at Scale.

Other ways to reduce costs include the following:

- Make sure to create alarms for the correct metrics.
- Make sure that you don't have any alarms enabled in Regions where you're not working.
- Remember that, although composite alarms reduce noise, they also generate additional costs.
- When deciding whether to create a standard alarm or high-resolution alarm, consider your use case and the value that each type of alarm brings.

Optimizing and reducing costs of CloudWatch Container Insights

CloudWatch Container Insights offers both standard and enhanced observability features for monitoring containerized applications in Amazon ECS and Amazon EKS. CloudWatch Container Insights leverages the embedded metric format to ingest telemetry from your container environments.

Container Insights with standard observability:

Standard Container Insights collects and visualizes metrics aggregated at the cluster and node levels. You can get started with standard mode of Container Insights using either the CloudWatch agent or the Amazon Distro for Open Telemetry (ADOT). Using ADOT enables you to customize which metrics and dimensions are sent to CloudWatch.

Metrics in Container Insights are treated as "embedded metrics". Costs associated with these metrics are reflected in the usage types `MetricStorage:AWS/Logs-EMF` and `DataProcessing-Bytes`. For detailed pricing information, see the Embedded Metrics section at [Amazon CloudWatch pricing].

Container Insights with enhanced observability:

Detailed visibility comes with Container Insights with enhanced observability, which delivers granular telemetry up to the pod and container levels in your applications. Similar to standard Container Insights, enhanced observability also comes with a standard set of critical metrics where you can get started by using the CloudWatch Observability add-on running on the CloudWatch agent. Container Insights offers enhanced observability with a new observation-based pricing in order to ensure cost effective bills justifying the benefit. See [Amazon CloudWatch pricing] for more information.

Here are the following UsageType and Operation associated with this Container Inisghts with enhanced observability:

CloudWatch sub-feature	UsageType	Operation
Container Insights with enhanced observability for Amazon EKS	`ObservationUsage`	`ObservationCount:CI-EKScode`
Container Insights with enhanced observability for Amazon ECS	`MetricsUsage`	`MetricStorage:CI-ECS`

Optimizing and reducing costs of CloudWatch Logs

Amazon CloudWatch Logs has the following log types:

- ***Custom logs*** *(logs that you create for your applications)*
- ***Vended logs*** *(logs that other Amazon Web Services services, such as Amazon Virtual Private Cloud (Amazon VPC) and Amazon Route 53, create on your behalf)*

For more information about vended logs, see Enabling logging from certain Amazon services in the *Amazon CloudWatch Logs User Guide*.

Custom and vended logs generate costs based on the number of logs that are *collected*, *stored*, and *analyzed*. Separately, vended logs generate costs for delivery to Amazon S3 and Firehose.

The following table lists the names of the CloudWatch Logs features and names of relevant sub-features. The table includes the strings for `UsageType` and `Operation`, which can help you analyze and identify log-related costs.

CloudWatch Logs feature	*CloudWatch Logs sub-feature*	UsageType	Operation	Purpose
Custom logs	*Collect (data ingestion for Standard log class)*	`DataProcessing-Bytes`	`PutLogEvents`	Uploads a batch of logs to a specific log stream in a

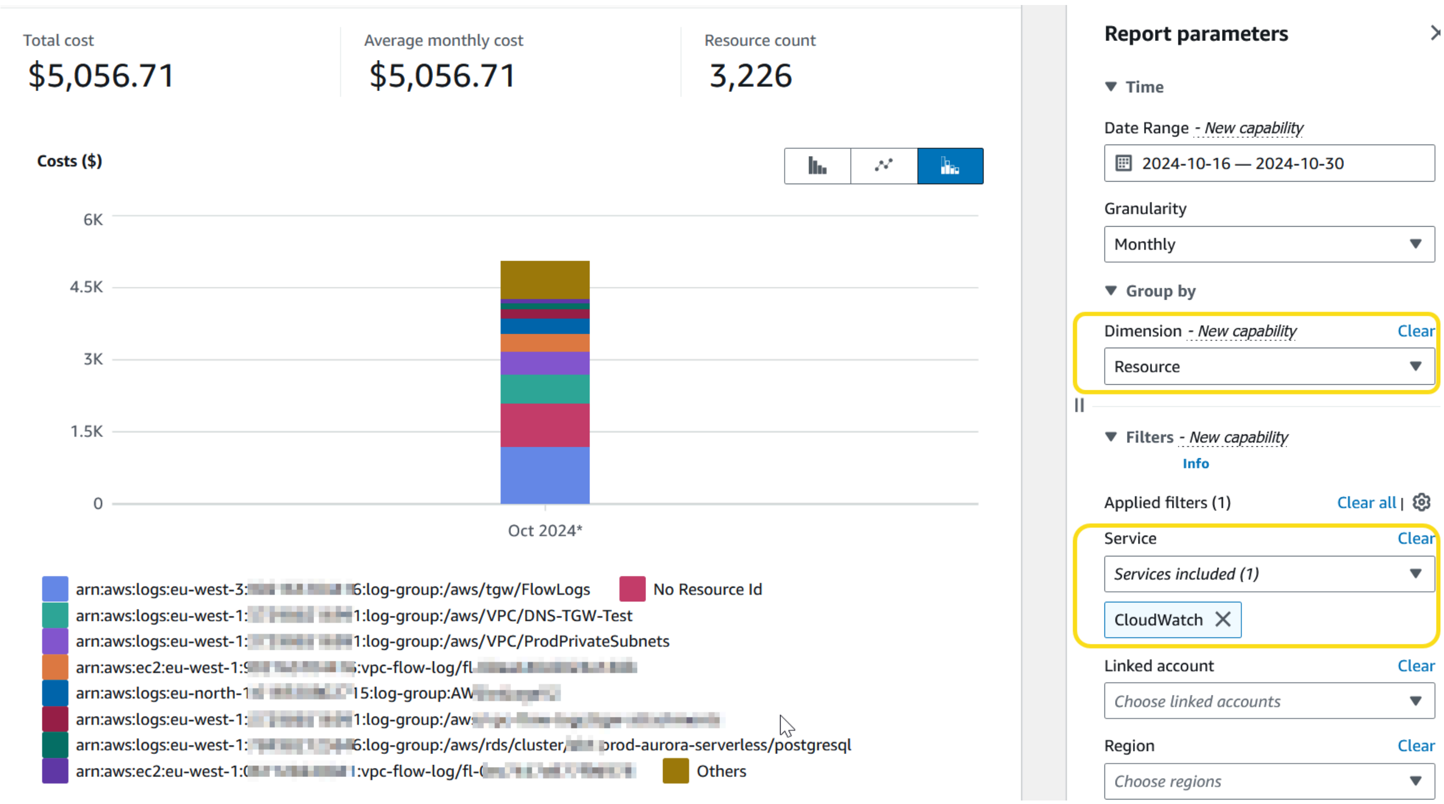
CloudWatch Logs feature	*CloudWatch Logs sub-feature*	UsageType	Operation	Purpose
				Standard class log group.
	Collect (data ingestion for Infrequent Access log class)	`DataProcessingIA-Bytes`	`PutLogEvents`	Uploads a batch of logs to a specific log stream in an Infrequent Access class log group.
	Detect and mask (data protection)	`DataProtection-Bytes`	`PutLogEvents`	Detects and masks protected data in log events.
	Store (archive)	`TimedStorage-ByteHrs`	`HourlyStorageMetering`	Stores logs per hour and logs per byte in CloudWatch Logs.
	Analyze (Logs Insights queries)	`DataScanned-Bytes`	`StartQuery`	Logs data scanned by CloudWatch Logs Insights queries
	Analyze (Logs Live Tail)	`Logs-LiveTail`	`StartLiveTail`	Logs analyzed during a CloudWatch Logs Live Tail session

CloudWatch Logs feature	*CloudWatch Logs sub-feature*	`UsageType`	`Operation`	Purpose
Vended logs	*Delivery (CloudWatch Logs Standard log class)*	`VendedLog-Bytes`	`PutLogEvents`	Uploads a batch of logs to a specific log stream in a log group in the Standard log class.
	Delivery (CloudWatch Logs Infrequent Access log class)	`VendedLogIA-Bytes`	`PutLogEvents`	Uploads a batch of logs to a specific log stream in a log group in the Infrequent Access log class.
	Delivery (Amazon S3)	`S3-Egress-Bytes`	`LogDelivery`	Uploads a batch of vended logs to a specific S3 bucket
	Delivery (Amazon S3) in Parquet format	`S3-Egress-InputBytes`	`ParquetConversion`	Perform Parquet conversion on logs delivered to Amazon S3
	Delivery (Firehose)	`FH-Egress-Bytes`	`LogDelivery`	Uploads a batch of vended logs to Amazon Data Firehose

To analyze costs, use Amazon Cost Explorer Service or Amazon Cost and Usage Reports with Athena. With either method, you can identify which logs are generating costs and determine how the costs are generated.

Using Amazon Cost Explorer Service

Select **CloudWatch** for the **Service** filter and select **Resource** for the **Dimension**. When you select **Resource** as the dimension in Cost Explorer Service you can see only the past 14 days of usage.



Using Amazon Athena query to track logs that generate costs

You can use the following query to track which logs generate costs by resource ID.

```
SELECT
line_item_resource_id AS ResourceID,
line_item_usage_type AS Operation,
SUM(CAST(line_item_unblended_cost AS decimal(16,8))) AS TotalSpend
FROM
costandusagereport
WHERE
product_product_name = 'AmazonCloudWatch'
AND year='2025'
AND month='1'
```

```
AND line_item_operation IN
 ('PutLogEvents','HourlyStorageMetering','StartQuery','LogDelivery','StartLiveTail','ParquetCon
AND line_item_line_item_type NOT IN
 ('Tax','Credit','Refund','EdpDiscount','Fee','RIFee')
GROUP BY
line_item_resource_id,
line_item_usage_type
ORDER BY
TotalSpend DESC
```

To make the most of costs that are generated by CloudWatch Logs, consider the following:

- Identify top log groups by spend per operation by using the previous query.
- Log only the events that bring your business value and choose an efficient log syntax. A verbose log syntax drives volume and thus cost. This helps you generate lower costs for ingestion.
- Change your log retention settings, so that you generate lower costs for storage. For more information, see Change log data retention in CloudWatch Logs in the *Amazon CloudWatch Logs User Guide*.
- Consider using the Infrequent Access log class where appropriate. Infrequent Access logs offers fewer features than Standard class. Determine whether you need the additional features of Standard log class and understand the difference between both classes. For more information, see the blog article New CloudWatch Logs log class for infrequent access logs at a reduced price. While the Infrequent Access class supports fewer features, it is suitable for a majority of use cases.
- Run queries that CloudWatch Logs Insights automatically saves in your history. This way, you generate fewer costs for analysis. For more information, see View running queries or query history in the *Amazon CloudWatch Logs User Guide*.
- Use the CloudWatch agent to collect system and application logs and send them to CloudWatch. This way, you can collect only the log events that meet your criteria. For more information, see Amazon CloudWatch Agent adds Support for Log Filter Expressions.

To reduce costs for vended logs, consider your use case, and then determine whether your logs should be sent to CloudWatch or Amazon S3. For more information, see Logs sent to Amazon S3 in the *Amazon CloudWatch Logs User Guide*.

> **Tip**
>
> If you want to use metric filters, subscription filters, CloudWatch Logs Insights, and Contributor Insights, send vended logs to CloudWatch.
> Alternatively, if you're working with VPC Flow Logs and using them for auditing and compliance purposes, send vended logs to Amazon S3.
> For information about how to track charges that are generated by publishing VPC Flow Logs to S3 buckets, see Using Amazon Cost and Usage Reports and cost allocation tags to understand VPC FLow Logs data ingestion in Amazon S3.

For additional information about how to make the most of costs that are generated by CloudWatch Logs, see Which log group is causing a sudden increase in my CloudWatch Logs bill?.

Using Amazon CloudWatch dashboards

Amazon CloudWatch includes automatic pre-built dashboards and also enables you to create your own dashboards. Dashboards help you to monitor your resources in a single view, even those resources that are spread across different Regions. You can use CloudWatch dashboards to create customized views of the telemetry data for your Amazon resources.

With customizable dashboards, you can create the following:

- A single view for selected metrics and alarms to help you assess the health of your resources and applications across one or more Regions. You can select the color used for each metric on each graph, so that you can easily track the same metric across multiple graphs.
- An operational playbook that provides guidance for team members during operational events about how to respond to specific incidents.
- A common view of critical resource and application measurements that can be shared by team members for faster communication flow during operational events.

If you have multiple Amazon accounts, you can set up *CloudWatch cross-account observability* and then create rich cross-account dashboards in your monitoring accounts. You can seamlessly search, visualize, and analyze your metrics, logs, and traces without account boundaries.

With CloudWatch cross-account observability, you can do the following in a dashboard in a monitoring account:

- Search, view, and create graphs of metrics that reside in source accounts. A single graph can include metrics from multiple accounts.
- Create alarms in the monitoring account that watch metrics in source accounts.
- View the log events from log groups located in source accounts, and run CloudWatch Logs Insights queries of log groups in source accounts. A single CloudWatch Logs Insights query in a monitoring account can query multiple log groups in multiple source accounts at once.
- View nodes from source accounts in a trace map in X-Ray. You can then filter the map to specific source accounts.

When you are signed in to a monitoring account, a blue **Monitoring account** badge appears at the top right of every page that supports CloudWatch cross-account observability functionality.

For more information about setting up CloudWatch cross-account observability, see CloudWatch cross-account observability.

You can create dashboards from the console or using the Amazon CLI or `PutDashboard` API operation. You can add dashboards to a favorites list, where you can access not only your favorite dashboards, but also your recently visited dashboards. For more information, see Add a dashboard to your favorites list.

To access CloudWatch dashboards, you need one of the following:

- The `AdministratorAccess` policy
- The `CloudWatchFullAccess` policy
- A custom policy that includes one or more of these specific permissions:
  - `cloudwatch:GetDashboard` and `cloudwatch:ListDashboards` to be able to view dashboards
  - `cloudwatch:PutDashboard` to be able to create or modify dashboards
  - `cloudwatch:DeleteDashboards` to be able to delete dashboards

Topics

Getting started with CloudWatch automatic dashboards

The CloudWatch home page automatically displays metrics about every Amazon service you use. You can additionally create custom dashboards to display metrics about your custom applications, and display custom collections of metrics that you choose.

Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.

The CloudWatch overview home page appears.

The overview displays the following items, refreshed automatically.

- **Alarms by Amazon service** displays a list of Amazon services you use in your account, along with the state of alarms in those services. Next to that, two or four alarms in your account are displayed. The number depends on how many Amazon services you use. The alarms shown are those in the ALARM state or those that most recently changed state.

  These upper areas help you quickly assess the health of your Amazon services, by seeing the alarm states in every service and the alarms that most recently changed state. This helps you monitor and quickly diagnose issues.
- Below these areas is the *default dashboard*, if one exists. The default dashboard is a custom dashboard that you have created and named **CloudWatch-Default**. This is a convenient way for you to add metrics about your own custom services or applications to the overview page, or to bring forward additional key metrics from Amazon services that you most want to monitor.

> **Note**
>
> The automatic dashboards on the CloudWatch home page display only information from the current account, even if the account is a monitoring account set up for CloudWatch cross-account observability. For information about creating custom cross-account dashboards, see Creating a CloudWatch cross-account cross-Region dashboard with the Amazon Web Services Management Console.

From this overview, you can see a cross-service dashboard of metrics from multiple Amazon service, or focus your view to a specific resource group or a specific Amazon service. This enables you to narrow your view to a subset of resources in which you are interested.

Topics

Viewing the cross-service CloudWatch dashboard

You can switch to the Cross-service dashboard screen and interact with dashboards for all of the Amazon services that you're using. The CloudWatch Console displays your dashboards in alphabetical order and displays one or two key metrics from each service.

> **Note**
>
> If you're using five or more Amazon services, the CloudWatch Console won't display the Cross-service dashboard on the Overview screen.

To view the Cross-service dashboard

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.

   You're directed to the Overview screen.

2. From the Overview screen, select the dropdown that reads **Overview**, and then choose **Cross service dashboard**.

   You're directed to the Cross service dashboard screen.

3. (Optional) If you're using the original interface, scroll to the section **Cross-service dashboard**, and then choose **View Cross-service dashboard**.

   You're directed to the Cross-service dashboard screen.

4. You can focus on a particular service in two ways:

   a. To see more key metrics for a service, choose its name from the list at the top of the screen, where **Cross service dashboard** is currently shown. Or, you can choose **View *Service* dashboard** next to the service name.

An automatic dashboard for that service is displayed, showing more metrics for that service. Additionally, for some services, the bottom of the service dashboard displays resources related to that service. You can choose one of those resources to that service console and focus further on that resource.

b. To see all the alarms related to a service, choose the button on the right of the screen next to that service name. The text on this button indicates how many alarms you have created in this service, and whether any are in the ALARM state.

   When the alarms are displayed, multiple alarms that have similar settings (such as dimensions, threshold, or period) may be shown in a single graph.

   You can then view details about an alarm and see the alarm history. To do so, hover on the alarm graph, and choose the actions icon, **View in alarms**.

   The alarms view appears in a new browser tab, displaying a list of your alarms, along with details about the chosen alarm. To see the history for this alarm, choose the **History** tab.

5. You can focus on resources in a particular resource group. To do so, choose the resource group from the list at the top of the page where **All resources** is displayed.

   For more information, see Viewing a CloudWatch dashboard for a resource group.

6. To change the time range shown in all graphs and alarms currently displayed, select the range you want next to **Time range** at the top of the screen. Choose **custom** to select from more time range options than those displayed by default.

7. Alarms are always refreshed once a minute. To refresh the view, choose the refresh icon (two curved arrows) at the top right of the screen. To change the automatic refresh rate for items on the screen other than alarms, choose the down arrow next to the refresh icon and choose the refresh rate you want. You can also choose to turn off automatic refresh.

Removing a service from appearing in the CloudWatch cross-service dashboard

You can prevent a service's metrics from appearing in the cross-service dashboard. This helps you focus your cross-service dashboard on the services you most want to monitor.

If you remove a service from the cross-service dashboard, the alarms for that service still appear in the views of your alarms.

To remove a service's metrics from the cross-service dashboard

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.

   The home page appears.

2. At the top of the page, under **Overview**, choose the service you want to remove.

   The view changes to show metrics from only that service.

3. Choose **Actions**, then clear the check box next to **Show on cross service dashboard**.

Viewing a CloudWatch dashboard for a single Amazon service

On the CloudWatch home page, you can focus the view to a single Amazon service. You can drill down further by focusing on both a single Amazon service and a resource group at the same time. The following procedure shows only how to focus on an Amazon service.

To focus on a single service

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.

   The home page appears.

2. For **Overview**, where **Overview** is currently shown in the drop down menu, choose **Service dashboards**.
3. Choose the service that you want to focus on.

   The view changes to display graphs of key metrics from the selected service.

4. To switch to viewing the alarms for this service, select the check box for **In alarm**, **Insufficient data**, or **OK** near the top of the screen where the service name is currently displayed.
5. When viewing metrics, you can focus on a particular metric in several ways:

   a. To see more details about the metrics in any graph, hover on the graph, and choose the actions icon, **View in metrics**.

      The graph appears in a new tab, with the relevant metrics listed below the graph. You can customize your view of this graph, changing the metrics and resources shown, the statistic, the period, and other factors to get a better understanding of the current situation.

b. You can view log events from the time range shown in the graph. This may help you discover events that happened in your infrastructure that are causing an unexpected change in your metrics.

   To see the log events, hover on the graph, and choose the actions icon, **View in logs**.

   The CloudWatch Logs view appears in a new tab, displaying a list of your log groups. To see the log events in one of these log groups that occurred during the time range shown in the original graph, choose that log group.

6. When viewing alarms, you can focus on a particular alarm in several ways:

   - To see more details about an alarm, hover on the alarm, and choose the actions icon, **View in alarms**.

     The alarms view appears in a new tab, displaying a list of your alarms, along with details about the chosen alarm. To see the history for this alarm, choose the **History** tab.

7. Alarms are always refreshed one time per minute. To refresh the view, choose the refresh icon (two curved arrows) at the top right of the screen. To change the automatic refresh rate for items on the screen other than alarms, choose the down arrow next to the refresh icon and choose a refresh rate. You can also choose to turn off automatic refresh.

8. To change the time range shown in all graphs and alarms currently displayed, next to **Time range** at the top of the screen, choose the range. To select from more time range options than those displayed by default, choose **custom**.

9. To return to the cross-service dashboard, choose **Overview** in the list at the top of the screen that currently shows the service you are focusing on.

   Alternatively, from any view, you can choose **CloudWatch** at the top of the screen to clear all filters and return to the overview page.

Viewing a CloudWatch dashboard for a resource group

You can focus your view to display metrics and alarms from a single resource group. Using resource groups enables you to use tags to organize projects, focus on a subset of your architecture, or distinguish between your production and development environments. They also enable you to focus on each of these resource groups on the CloudWatch overview. For more information, see What Is Amazon Resource Groups?.

When you focus on a resource group, the display changes to show only the services where you have tagged resources as part of this resource group. The recent alarms area shows only the alarms that are associated with resources that are part of the resource group. Additionally, if you have created a dashboard with the name **CloudWatch-Default-*ResourceGroupName***, it is displayed in the **Default dashboard** area.

You can drill down further by focusing on both a single Amazon service and a resource group at the same time. The following procedure shows just how to focus on a resource group.

To focus on a single resource group

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. At the top of the page, where **All resources** is displayed, choose a resource group.
3. To see more metrics related to this resource group, near the bottom of the screen, choose **View cross service dashboard**.

   The cross-service dashboard appears, showing only the services related to this resource group. For each service, one or two key metrics are displayed.
4. To change the time range shown in all graphs and alarms currently displayed, for **Time range** at the top of the screen, select a range. To select from more time range options than those displayed by default, choose **custom**.
5. Alarms are always refreshed one time per minute. To refresh the view, choose the refresh icon (two curved arrows) at the top right of the screen. To change the automatic refresh rate for items on the screen other than alarms, choose the down arrow next to the refresh icon and choose a refresh rate. You can also choose to turn off automatic refresh.
6. To return to showing information about all the resources in your account, near the top of the screen where the name of the resource group is currently displayed, choose **All resources**.

Creating a customized CloudWatch dashboard

To get started, create a CloudWatch dashboard. You can create multiple dashboards, and you can add dashboards to a favorites list. You aren't limited to the number of dashboards that you can have in your Amazon Web Services account. All dashboards are global. They are not Region-specific.

The following procedure shows you how to create a dashboard from the CloudWatch console. You can use the `PutDashboard` API operation to create a dashboard from the command line interface. The API operation contains a JSON string that defines your dashboard content. For more information about creating a dashboard with the `PutDashboard` API operation, see PutDashboard in the *Amazon CloudWatch API Reference*.

> **Tip**
>
> If you're creating a new dashboard with the `PutDashboard` API operation, you can use the JSON string from a dashboard that already exists.

To create a dashboard from the console

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose **Create dashboard**.
3. In the **Create new dashboard** dialog box, enter a name for the dashboard, and then choose **Create dashboard**.

   If you use the name **CloudWatch-Default** or **CloudWatch-Default-*`ResourceGroupName`***, the dashboard appears in the overview of the CloudWatch home page under **Default Dashboard**. For more information, see Getting started with CloudWatch automatic dashboards.
4. In the **Add to this dashboard** dialog box, do one of the following:
   - To add a graph to the dashboard, choose **Line** or **Stacked area**, and then choose **Configure**. In the **Add metric graph** dialog box, select the metric(s) to graph, and then choose **Create widget**. If a metric doesn't appear in the dialog box because it hasn't published data in more than 14 days, you can add it manually. For more information, see Graph metrics manually on a CloudWatch dashboard.
   - To add a number displaying a metric to the dashboard, choose **Number**, and then choose **Configure**. In the **Add metric graph** dialog box, select the metric(s) to graph, and then choose **Create widget**.
   - To add a text block to the dashboard, choose **Text**, and then choose **Configure**. In the **New text widget** dialog box, for **Markdown**, format your text using Markdown, and then choose **Create widget**.
5. (Optional) Choose **Add widget**, and then repeat step 4 to add another widget to the dashboard. You can repeat this step multiple times.

   For each graph on the dashboard, there is an information icon at the upper right. Choose this icon to see the descriptions of the metrics in the graph.

6. Choose **Save dashboard**.

Creating a CloudWatch cross-account cross-Region dashboard with the Amazon Web Services Management Console

You can create *cross-account cross-Region dashboards*, which summarize your CloudWatch data from multiple Amazon accounts and multiple Regions into one dashboard. From this high-level dashboard you can get a view of your entire application, and also drill down into more specific dashboards without having to sign in and out of accounts or switch Regions.

Prerequisite

Before you can create a cross-account cross-Region dashboard, you must enable at least one sharing account and at least one monitoring account. Additionally, to be able to use the CloudWatch console to create a cross-account dashboard, you must enable the console for cross-account functionality. For more information, see Cross-account cross-Region CloudWatch console.

To create a cross-account cross-Region dashboard

1. Sign in to the monitoring account.
2. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
3. In the navigation pane, choose **Dashboards**.
4. Choose a dashboard, or create a new one.
5. At the top of the screen, you can switch between accounts and Regions. As you create your dashboard, you can include widgets from multiple accounts and Regions. Widgets include graphs, alarms, and CloudWatch Logs Insights widgets.

Create a cross-account cross-Region dashboard programmatically

You can use the Amazon APIs and SDKs to create dashboards programmatically. For more information, see PutDashboard.

To enable cross-account cross-Region dashboards, we have added new parameters to the dashboard body structure, as shown in the following table and examples. For more information about overall dashboard body structure, see Dashboard Body Structure and Syntax.

Parameter	Use	Scope	Default
accountId	Specifies the ID of the account where the widget or the metric is located.	Widget or metric	Account that is currently logged in
region	Specifies the Region of the metric.	Widget or metric	Current Region selected in the console

The following examples illustrate the JSON source for widgets in a cross-account cross-Region dashboard.

This example sets the accountId field to the ID of the sharing account at the widget level. This specifies that all metrics in this widget will come from that sharing account and Region.

```
{
  "widgets": [
    {
          ...
          "properties": {
        "metrics": [
                 …
        ],
        "accountId": "111122223333",
        "region": "us-east-1"
      }
    }
  ]
}
```

This example sets the accountId field differently at the level of each metric. In this example, the different metrics in this metric math expression come from different sharing accounts and different Regions.

```
{
```

```
  "widgets": [
    {
          ...
          "properties": {
        "metrics": [
          [
            {
              "expression": "SUM(METRICS())",
              "label": "[avg: ${AVG}] Expression1",
              "id": "e1",
              "stat": "Sum"
            }
          ],
          [
            "AWS/EC2",
            "CPUUtilization",
            {
              "id": "m2",
              "accountId": "5555666677778888",
              "region": "us-east-1",
              "label": "[avg: ${AVG}] ApplicationALabel "
            }
          ],
          [
            ".",
            ".",
            {
              "id": "m1",
              "accountId": "9999000011112222",
              "region": "eu-west-1",
              "label": "[avg: ${AVG}] ApplicationBLabel"
            }
          ]
        ],
        "view": "timeSeries",
        "region": "us-east-1", ---> home region of the metric. Not present in above
 example
              "stacked": false,
        "stat": "Sum",
        "period": 300,
        "title": "Cross account example"
      }
    }
  ]
```

```
}
```

This example shows an alarm widget.

```
{
  "type": "metric",
  "x": 6,
  "y": 0,
  "width": 6,
  "height": 6,
  "properties": {
    "accountID": "111122223333",
    "title": "over50",
    "annotations": {
      "alarms": [
        "arn:aws:cloudwatch:us-east-1:379642911888:alarm:over50"
      ]
    },
    "view": "timeSeries",
    "stacked": false
  }
}
```

This example is for a CloudWatch Logs Insights widget.

```
{
  "type": "log",
  "x": 0,
  "y": 6,
  "width": 24,
  "height": 6,
  "properties": {
    "query": "SOURCE 'route53test' | fields @timestamp, @message\n| sort @timestamp
 desc\n| limit 20",
    "accountId": "111122223333",
    "region": "us-east-1",
    "stacked": false,
```

```
    "view": "table"
  }
}
```

Another way to create dashboards programmatically is to first create one in the Amazon Web Services Management Console, and then copy the JSON source of this dashboard. To do so, load the dashboard and choose **Actions**, **View/edit source**. You can then copy this dashboard JSON to use as a template to create similar dashboards.

Creating a graph with metrics from different accounts and Regions in a CloudWatch dashboard

1. Sign in to the monitoring account.
2. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
3. In the navigation pane, choose **Metrics**, and then choose **All metrics**.
4. Select the account and Region that you want to add metrics from. You can select your account and Region from the account and Region dropdown menus near the top-right of the screen.
5. Add the metrics you want to the graph. For more information, see Graphing metrics.
6. Repeat steps 4-5 to add metrics from other accounts and Regions.
7. (Optional) Choose the **Graphed metrics** tab and add a metric math function that uses the metrics that you have chosen. For more information, see Using math expressions with CloudWatch metrics.

   You can also set up a single graph to include multiple SEARCH functions. Each search can refer to a different account or Region.
8. When you are finished with the graph, choose **Actions**, **Add to dashboard**.

   Select your cross-account dashboard, and choose **Add to dashboard**.

Adding an alarm from a different account to a CloudWatch cross-account dashboard

1. Sign in to the monitoring account.

2. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
3. At the top of the page, choose the account where the alarm is located.
4. In the navigation pane, choose **Alarms**.
5. Select the check box next to the alarm that you want to add, and choose **Add to dashboard**.
6. Select the cross-account dashboard that you want to add it to, and choose **Add to dashboard**.

Creating flexible CloudWatch dashboards with dashboard variables

Use *dashboard variables* to create flexible dashboards that can quickly display different content in multiple widgets, depending on the value of an input field within the dashboard. For example, you can create a dashboard that can quickly switch between different Lambda functions or Amazon EC2 instance IDs, or one that can switch to different Amazon Regions.

After you create a dashboard that uses a variable, you can copy the same variable pattern to other existing dashboards.

Using dashboard variables improves the operational workflow for people who use your dashboards. It can also reduce your costs because you're using dashboard variables in one dashboard instead of creating multiple similar dashboards.

Note

If you share a dashboard that contains dashboard variables, the people that you share it with won't be able to change between the variable values.

Types of dashboard variables

The dashboard variable can be a *property variable* or a *pattern variable*.

- *Property variables* change all instances of a property in all widgets in the dashboard. This property can be any JSON property in the JSON source of a dashboard, such as `region`. Or it can be a dimension name for a metric, such as `InstanceID` or `FunctionName`.

  For a tutorial that uses a property variable, see [Tutorial: Creating a CloudWatch Lambda dashboard with function name as the variable](#).

For more information about the JSON source of dashboards, see Dashboard Body Structure and Syntax. In the CloudWatch console, you can see the JSON source for any custom dashboard by choosing **Actions**, **View/edit source**.

- *Pattern variables* use a regular expression pattern to change all of a JSON property or only a certain part of it.

  For a tutorial that uses a pattern variable, see Tutorial: Creating a dashboard that uses a regular expression pattern to switch between Amazon Web Services Regions.

Property variables apply to most use cases and are less complex to set up.

Topics

- Copying a variable to another CloudWatch dashboard
- Tutorial: Creating a dashboard that uses a regular expression pattern to switch between Amazon Web Services Regions
- Tutorial: Creating a CloudWatch Lambda dashboard with function name as the variable

Copying a variable to another CloudWatch dashboard

After you create a dashboard with useful variables, you can copy these variables to other existing dashboards. For more information about dashboard variables, see Creating flexible CloudWatch dashboards with dashboard variables.

To copy a dashboard variable to another dashboard

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose the name of the dashboard that has the variable that you want to copy. Enter a string to find dashboards with matching names, if needed.
3. Choose **Actions**, **Variables**, **Manage variables**.
4. Choose the radio button next to the variable that you want to copy, and choose **Copy to another dashboard**.
5. Choose the selection box and start typing the dashboard name that you want to copy the variable to.
6. Select the dashboard name and choose **Copy variable**.

Tutorial: Creating a dashboard that uses a regular expression pattern to switch between Amazon Web Services Regions

The steps in this procedure illustrate how to create a flexible dashboard that can switch between Regions. This tutorial uses a regular expression *pattern variable* instead of a property variable. For a tutorial that uses a property variable, see Tutorial: Creating a CloudWatch Lambda dashboard with function name as the variable.

For many use cases, you can create a dashboard that switches between Regions by using a property variable. But if the widgets rely on Amazon Resource Names (ARNs) that include Region names, you must use a pattern variable to change the Region names within the ARNs.

To use a dashboard pattern variable to create a flexible multi-Region dashboard

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, **Create dashboard**.
3. Enter a name for the dashboard, and choose **Create dashboard**.
4. Add widgets to the dashboard. When you add the widgets that you want to display Region-specific data, avoid specifying any dimensions with values that appear in only one Region. For example, for Amazon EC2 metrics, specify metrics that are aggregated instead of metrics that use **InstanceID** as a dimension.

   For more information about adding widgets to a dashboard, see Using widgets on CloudWatch dashboards.
5. After you add the widgets, as you are viewing the dashboard, choose **Actions**, **Variables**, **Create a variable**.
6. Choose **Pattern variable**.
7. For **Property that the variable changes**, enter the name of the current dashboard Region, such as **`us-east-2`**.

   You have the correct Region entered if the label below that box displays the widgets that will be impacted by the variable.
8. For **Input type**, for this use case, select **Radio button**.
9. For **Define how inputs are populated**, choose **Create a list of custom values**.

10. For **Create your custom values**, enter the Regions that you want to switch between, with one Region on each line. After each Region, enter a comma and then the label to display for that radio button. For example:

    `us-east-1, N. Virginia`

    `us-east-2, Ohio`

    `eu-west-3, Paris`

    As you fill in the custom values, the **Preview** pane updates to display what the radio buttons will look like.

11. (Optional) For more settings, choose **Secondary settings** and do one or more of the following:

    - To customize the name of your variable, enter the name in **Custom variable name**.
    - To customize the label for the variable input field, enter the label in **Input label**. For this tutorial, enter **`Region:`**.

      If you enter a value here, the **Preview** pane updates to display what the radio buttons will look like.
    - To set the default value for this variable when the dashboard is first opened, enter the default in **Default value**.

12. Choose **Add variable**.

    The dashboard appears, with a **Region:** label next to the radio buttons for the Regions near the top. When you switch between Regions, all the widgets that use the variable will display information about the selected Region.

Tutorial: Creating a CloudWatch Lambda dashboard with function name as the variable

The steps in this procedure illustrate how to create a flexible dashboard that shows a variety of metric graphs, using a property variable. This includes a dropdown selection box on the dashboard that you can use to switch the metrics in all the graphs between different Lambda functions.

Other use case examples for this type of dashboard include using `InstanceId` as the variable to create a dashboard of metrics with a dropdown for instance IDs. Alternatively, you could create a dashboard that uses `region` as the variable to display the same set of metrics from different Regions.

To use a dashboard property variable to create a flexible Lambda dashboard

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, **Create dashboard**.
3. Enter a name for the dashboard, and choose **Create dashboard**.
4. Add widgets to the dashboard that display metrics for a Lambda function. When you create these widgets, specify **Lambda**, **By Function Name** for the widget metrics. For the function, specify one of the Lambda functions that you want to include in this dashboard.

   For more information about adding widgets to a dashboard, see Using widgets on CloudWatch dashboards.
5. After you add the widgets, as you are viewing the dashboard, choose **Actions**, **Variables**, **Create a variable**.
6. Choose **Property variable**.
7. For **Property that the variable changes**, choose **FunctionName**.
8. For **Input type**, for this use case, we recommend choosing **Select menu (dropdown)**. This creates a dropdown menu in the dashboard where you can select the Lambda function name to display metrics for.

   If this was for a dashboard that toggled between only two or three different values for a variable, then **Radio button** would be a good choice.

   If you prefer to enter or paste in values for the variable, you would choose **Text input**. This option doesn't include a dropdown list or radio buttons.
9. When you choose **Select menu (dropdown)**, you must then choose whether to populate the menu by entering values, or using a metric search. For this use case, let's assume that you have a large number of Lambda functions and you don't want to enter all of them manually. Choose **Use the results of a metric search** and then do the following:

   a. Choose **Pre-built queries**, **Lambda**, **Errors**.

      (Choosing **Errors** does not add the **Errors** metric to the dashboard. However, it quickly populates the **FunctionName** variable selection box.)
   b. Choose **By Function Name** and then choose **Search**.

Under the **Search** button, you will then see **FunctionName** selected. You also see a message about how many **FunctionName** dimension values were found to populate the input box.

10. (Optional) For more settings, choose **Secondary settings** and do one or more of the following:
    - To customize the name of your variable, enter the name in **Custom variable name**.
    - To customize the label for the variable input field, enter the label in **Input label**.
    - To set the default value for this variable when the dashboard is first opened, enter the default in **Default value**.
11. Choose **Add variable**.

    A **FunctionName** dropdown selection box appears near the top of the dashboard. You can select a Lambda function in this box and all the widgets that use the variable will display information about the selected function.

    Later, if you add more widgets to the dashboard that watch Lambda metrics with the **FunctionName** dimension, they will automatically use the variable.

Using widgets on CloudWatch dashboards

You can use the following types of widgets on CloudWatch dashboards:

- **Graph widget** – You can add graphs that contain one or more metrics to your CloudWatch dashboard. The types of graphs that you can add to your dashboard include ***Line***, ***Stacked area***, ***Number***, ***Gauge***, ***Bar***, and ***Pie***. You can remove graphs from your dashboard when you don't need them anymore.
- **Manual metrics widget** – If a metric hasn't published data in the past 14 days, you can't find it when searching for metrics to add to a graph on a CloudWatch dashboard. You can add any metric manually to an existing graph.
- **Metrics explorer widget –** Metrics explorer widgets include graphs of multiple resources that have the same tag, or share the same resource property such as an instance type. These widgets stay up to date, as resources that match are created or deleted. Adding metrics explorer widgets to your dashboard helps you to troubleshoot your environment more efficiently.

  For example, you can monitor your fleet of EC2 instances by assigning tags that represent their environments, such as production or test. You can then use these tags to filter and aggregate the operational metrics, such as `CPUUtilization`, to understand the health and performance of the EC2 instances that are associated with each tag.

- **Line widget** – With the line widget, you can compare metrics over periods of time. You also can use the widget's mini-map zoom feature to inspect sections of line graphs without changing between zoomed-in and zoomed-out views. The procedures in this section describe how to add and remove a line widget on a CloudWatch dashboard. For information about using the widget's mini-map zoom feature with line graphs, see Zooming in on a line or stacked area graph.
- **Number widget** – With the number widget, you can look at the latest metric values and trends as soon as they appear. Because the number widget includes the sparkline feature, you can visualize the top and bottom halves of metric trends in a single graph. The procedures in this section describe how to add and remove a number widget from a CloudWatch dashboard.
- **Gauge widget** – With the gauge widget, you can visualize metric values that go between ranges. For example, you can use the gauge widget to graph percentages and CPU utilization, so that you can observe and diagnose any performance issues that occur. The procedures in this section describe how to add and remove a gauge widget from a CloudWatch dashboard.
- **Text widget** – A text widget contains a block of text in Markdown format. You can add, edit, or remove text widgets from your CloudWatch dashboard.
- **Alarm widget** – To add an alarm widget to a dashboard, choose one of the following options:
  - Add a single alarm in a widget, which displays the graph of the alarm's metric and also displays the alarm status.
  - Add an *alarm status widget*, which displays the status of multiple alarms in a grid. Only the alarm names and current status are displayed, Graphs are not displayed. You can include up to 100 alarms in one alarm status widget.
- **Table widget** – With the data table widget, you can see the raw datapoints of your metric and a quick summary of that raw data. Because the data table widget is not a chart to abstract the actual data away from you, it is easier to understand the datapoints being presented. The procedures in this section describe how to add and remove a data table widget from a CloudWatch dashboard. The following image shows an example of a table widget with columns for the Minimum, Maximum, Sum, and Average statistics for a set of CloudWatch metrics.
- **Linked graphs** – You can link the graphs on your dashboard together, so that when you zoom in or zoom out on one graph, the other graphs zoom in or zoom out at the same time. You can unlink graphs to limit zoom to one graph.

Topics

Adding a graph widget to a CloudWatch dashboard

The procedures in this section describe how to add and remove graphs from your dashboard. For information about how to edit a graph on your dashboard, see Edit a graph on a CloudWatch dashboard.

To add a graph to a dashboard

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. Choose the **+** symbol, and then choose the type of graph that you want to add to your dashboard, then choose **Next**.
   - If you select ***Line***, ***Stacked area***, ***Bar***, or ***Pie***, choose **Metrics**.
4. In the **Browse** tab, search or browse for the metrics to graph, and select the ones that you want.
5. (Optional) To change your graph's time range, select one of the predefined time ranges in the upper part of the screen. The time ranges span from 1 hour to 1 week (***1h***, ***3h***, ***12h***, ***1d***, ***3d***, or ***1w***).

   To set your own time range, choose **Custom**.
   - (Optional) To have this widget keep using this time range that you select, even if the time range for the rest of the dashboard is later changed, choose **Persist time range**.
6. (Optional) To change your graph's widget type, use the dropdown that's next to the predefined time ranges.
7. (Optional) In **Graphed metrics**, you can add a dynamic label to your metric and change your metric's label, label color, statistic, and period. You also can determine the position of labels on the Y-axis from left to right.

   a. To add a dynamic label, choose **Graphed metrics**, and then choose **Add dynamic labels**. Dynamic labels display a statistic about your metric in the graph legend. Dynamic labels update automatically when your dashboard or graph refreshes. By default, the dynamic values that you add to labels appear at the beginning of your labels. For more information, see Use dynamic labels.

   b. To change the color of a metric, choose the color square that's next to the metric.

   c. To change the statistic, select the dropdown under ***Statistic***, and then choose a new value. For more information, see Statistics.

   d. To change the period, select the dropdown under the ***Period*** column, and then choose a new value.
8. If you are creating a gauge widget, you must choose the **Options** tab and specify the **Min** and **Max** values to use for the two ends of the gauge.

9. (Optional) To customize the Y-axis, choose **Options**. You can add a custom label under ***Left Y-axis*** in the label field. If your graph displays values on the right side of the Y-axis, you can customize that label, too. You also can set minimum and maximum limits on your Y-axis values, so that your graph only displays the value ranges that you specify.

10. (Optional) To add or edit horizontal annotations to line or stacked area graphs, or to add thresholds to gauge widgets, choose **Options**:

    a. To add a horizontal annotation or threshold, choose **Add horizontal annotation** or **Add threshold**.

    b. For ***Label***, enter a label for the annotation then choose the check mark icon.

    c. For ***Value***, choose the pen and paper icon that's next to the current value, and enter your new value. After you enter your value, choose the check mark icon.

    d. For ***Fill***, select the dropdown and specify how your annotation will use shading. You can choose ***None***, ***Above***, ***Between***, or ***Below***. To change the fill color, choose the color square that's next to the annotation.

    e. For ***Axis***, specify whether your annotation appears on the left or right side of the Y-axis.

    f. To hide an annotation, clear the check box that's next to the annotation you want to hide.

    g. To delete an annotation, choose **X** under ***Actions***.

> **Note**
>
> You can repeat these steps to add multiple horizontal annotations or thresholds to the same graph or gauge.

11. (Optional) To add or edit vertical annotations, choose **Options**:

    a. To add a vertical annotation, choose **Add vertical annotation**.

    b. For ***Label***, choose the pen and paper icon that's next to the current annotation, and enter your new annotation. If you want to show only the date and time, leave the label field blank.

    c. For ***Date***, choose the current date and time, and enter the new date and time.

    d. For ***Fill***, select the dropdown, and specify how your annotation will use shading. You can choose ***None***, ***Above***, ***Between***, or ***Below***. To change the fill color, select the color square that's next to the annotation.

    e. To hide an annotation, clear the check box next to the annotation that you want to hide.

f. To delete an annotation, choose **X** under ***Actions***.

> **Note**
>
> You can repeat these steps to add multiple vertical annotations to the same graph.

12. Choose **Create widget**.
13. Choose **Save dashboard**.

Removing a graph widget from a CloudWatch dashboard

To remove a graph from a dashboard

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. In the upper-right corner of the graph that you want to remove, choose **Widget actions**, and then choose **Delete**.
4. Choose **Save dashboard**.

Graph metrics manually on a CloudWatch dashboard

To add a metric that you can't find in search to a graph

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. The dashboard must already contain a graph where you want to add the metric. If it doesn't, create the graph and add any metric to it. For more information, see Adding a graph widget to a CloudWatch dashboard .
4. Choose **Actions**, **View/edit source**.

   A JSON block appears. The block specifies the widgets on the dashboard and their contents. The following is an example of one part of this block, which defines one graph.

```
{
```

```
  "type": "metric",
  "x": 0,
  "y": 0,
  "width": 6,
  "height": 3,
  "properties": {
    "view": "singleValue",
    "metrics": [
      [
        "AWS/EBS",
        "VolumeReadOps",
        "VolumeId",
        "vol-1234567890abcdef0"
      ]
    ],
    "region": "us-west-1"
  }
},
```

In this example, the following section defines the metric shown on this graph.

```
[ "AWS/EBS", "VolumeReadOps", "VolumeId", "vol-1234567890abcdef0" ]
```

5. Add a comma after the end bracket if there isn't already one and then add a similar bracketed section after the comma. In this new section, specify the namespace, metric name, and any necessary dimensions of the metric that you're adding to the graph. The following is an example.

```
[ "AWS/EBS", "VolumeReadOps", "VolumeId", "vol-1234567890abcdef0" ],
[ "MyNamespace", "MyMetricName", "DimensionName", "DimensionValue" ]
```

For more information about the formatting of metrics in JSON, see [Properties of a Metric Widget Object](#) in the Amazon CloudWatch API Reference.

6. Choose **Update**.

Editing a graph on a CloudWatch dashboard

You can edit the graphs that you add to your CloudWatch dashboard. You can change a graph's title, statistic, or period. You can add, update, and remove metrics from your graphs. If your graph contains more than one metric, you can reduce clutter by hiding metrics that you aren't using. The procedures in this section describe how to edit a graph on your dashboard. For information about creating a graph, see [Add or remove a graph from a CloudWatch dashboard](#).

New interface

To edit a graph on a dashboard

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. In the upper-right corner of the graph that you want to edit, choose **Widget actions**, and then choose **Edit**.
4. To change the graph's title, choose the pen and paper icon that's next to the current title. Enter the new title, and then choose **Apply**.
5. (Optional) To change your graph's time range, select one of the predefined time ranges in the upper are of the graph. The time ranges span from 1 hour to 1 week (***1h***, ***3h***, ***12h***, ***1d***, ***3d***, or ***1w***).

   To set your own time range, choose **Custom**.

   - (Optional) To have this widget keep using this time range that you select, even if the time range for the rest of the dashboard is later changed, choose **Persist time range**.
6. To change your graph's widget type, use dropdown that's next to the predefined time ranges.
7. In **Graphed metrics**, you can add a dynamic label to your metric and change your metric's label, label color, statistic, and period. You also can determine the position of labels on the Y-axis from left to right.

   a. To add a dynamic label for a metric, choose **Dynamic labels**. Dynamic labels display a statistic about the metric in the graph legend. Dynamic labels update automatically when your dashboard or graph refreshes. By default, the dynamic values that you add to labels appear at the beginning of the labels. For more information, see [Use dynamic labels](#).

b. To change the color of a metric, choose the color square that's next to the metric.

c. To change the statistic, choose the statistic value under the ***Statistic*** column, and then choose a new value. For more information, see Statistics.

d. To change the period, choose the period value under the ***Period*** column, and then choose a new value.

8. To add or edit horizontal annotations, choose **Options**:

   a. To add a horizontal annotation, choose **Add horizontal annotation**.

   b. For **Label**, choose the pen and paper icon next to the current annotation. Then enter your new annotation. After you enter your annotation, choose the check mark icon.

   c. For **Value**, choose the pen and paper icon next to the current metric value. Then enter your new metric value. After you enter your value, select the check mark icon.

   d. For **Fill**, choose the dropdown under the column, and then specify how your annotation will use shading. You can choose **None**, **Above**, **Between**, or **Below**. If you choose **Between**, another new label and value field appears.

   > **Tip**
   >
   > You can change the fill color by choosing the colored square next to the annotation.

   e. For **Axis**, specify whether your annotation appears on the left or right side of the Y-axis.

   f. To hide an annotation, deselect the check box next to the annotation that you want to hide on the graph.

   g. To delete an annotation, choose **X** under the **Actions** column.

   > **Note**
   >
   > You can repeat these steps to add multiple horizontal annotations to the same graph.

9. To add or edit vertical annotations, choose **Options**:

   a. To add a vertical annotation, choose **Add vertical annotation**.

b. For **Label**, choose the pen and paper icon next to the current annotation. Then enter your new annotation. After you enter your annotation, choose the check mark icon.

> **Tip**
>
> To show only the date and time, leave the label field blank.

c. For **Date**, choose the current date and time. Then enter the new date and time.

d. For **Fill**, choose the dropdown under the column, and then specify how your annotation will use shading. You can choose **None**, **Above**, **Between**, or **Below**. If you choose **Between**, a new label and value field appears.

> **Tip**
>
> You can change the fill color by choosing the color square next to the annotation.

> **Note**
>
> You can repeat these steps to add multiple vertical annotations to the same graph.

e. To hide an annotation, deselect the check box next to the annotation that you want to hide on the graph.

f. To delete an annotation, choose **X** under the **Actions** column.

10. To customize the Y-axis, choose **Options**. Under ***Left Y-axis***, you can enter a custom label for **Label**. If the graph displays values on the right Y-axis, you can customize that label, too. You also can set minimums and maximums on the Y-axis values, so that the graph displays only the value range that you specify.
11. When you finish making changes, choose **Update widget**.

To hide or change the position of a graph legend

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.

3. In the upper-right corner of the graph that you want to edit, choose **Widget actions**. Choose **Legend** and select **Hidden**, **Bottom**, or **Right**.

To temporarily hide metrics for a graph on a dashboard

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. Select the color square for the metric that you want to hide in the graph's footer. An ***X*** appears in the color square when you hover over it, and the square turns gray when you choose it.
4. To restore the hidden metric, clear the **X** in the gray square.

Original interface

To edit a graph on a dashboard

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. Hover over the upper-right corner of the graph that you want to edit. Choose **Widget actions**, and then choose **Edit**.
4. To change the graph's title, choose the pencil icon that's next to the current title, and then enter the new title.
5. To change the graph's time range, choose one of the predefined time ranges in the upper area of the graph. These span from 1 hour to 1 week (***1h***, ***3h***, ***12h***, ***1d***, ***3d***, or ***1w***).
   - To set your own time range, choose **custom**.
6. To change your graph's widget type, select the **Graph options** tab. You can choose ***Line***, ***Stacked area***, ***Number***, ***Bar***, or ***Pie***.

   > **Tip**
   >
   > You can change your graph's widget type by choosing the dropdown that's next to the predefined time ranges.

7. In **Graphed metrics**, you can add a dynamic label to your metric and change your metric's label, label color, statistic, and period. You also can determine the position of labels on the Y-axis from left to right.

    a. To add a dynamic label for a metric, choose **Dynamic labels**. Dynamic labels display a statistic about the metric in the graph legend. Dynamic labels update automatically when your dashboard or graph refreshes. By default, the dynamic values that you add to labels appear at the beginning of the labels. For more information, see Use dynamic labels.

    b. To change the color of a metric, choose the color square that's next to the metric.

    c. To change the statistic, choose the statistic value under the ***Statistic*** column, and then choose a new value. For more information, see Statistics.

    d. To change the period, choose the period value under the ***Period*** column, and then choose a new value.

8. To add or edit horizontal annotations, choose **Graph options**:

    a. To add a horizontal annotation, choose **Add horizontal annotation**.

    b. For **Label**, choose the pencil icon next to the current annotation. Then enter your new annotation. After you enter your annotation, choose the check mark icon.

    c. For **Value**, choose the pencil icon next to the current metric value. Then enter your new metric value. After you enter your value, select the check mark icon.

    d. For **Fill**, choose the dropdown under the column, and then specify how your annotation will use shading. You can choose **None**, **Above**, **Between**, or **Below**. If you choose **Between**, a new label and value field appears.

    > **Tip**
    >
    > You can change the fill color by choosing the color square next to the annotation.

    e. For **Axis**, specify whether your annotation appears on the left or right side of the Y-axis.

    f. To hide an annotation, deselect the check box next to the annotation that you want to hide on the graph.

    g. To delete an annotation, choose **X** under the **Actions** column.

> **Note**
>
> You can repeat these steps to add multiple horizontal annotations to the same graph.

9. To add or edit vertical annotations, choose **Graph options**:

   a. To add a vertical annotation, choose **Add vertical annotation**.

   b. For **Label**, choose the pencil icon next to the current annotation. Then enter your new annotation. After you enter your annotation, choose the check mark icon.

   > **Tip**
   >
   > To show only the date and time, leave the label field blank.

   c. For **Date**, choose the pencil icon next to the current date and time. Then enter the new date and time.

   d. For **Fill**, choose the dropdown under the column, and then specify how your annotation will use shading. You can choose **None**, **Above**, **Between**, or **Below**. If you choose **Between**, a new label and value field appears.

   > **Tip**
   >
   > You can change the fill color by choosing the color square next to the annotation.

   > **Note**
   >
   > You can repeat these steps to add multiple vertical annotations to the same graph.

   e. To hide an annotation, deselect the check box next to the annotation that you want to hide on the graph.

   f. To delete an annotation, choose **X** under the **Actions** column.

10. To customize the Y-axis, choose **Graph options**. Under ***Left Y-axis***, you can enter a custom label for **Label**. If the graph displays values on the right Y-axis, you can customize that label, too. You also can set minimums and maximums on the Y-axis values, so that the graph displays only the value range that you specify.
11. When you finish making changes, choose **Update widget**.

To hide or change the position of a graph legend

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. Hover over the upper-right corner of the graph that you want to edit, and choose **Widget actions**. Choose **Legend**, and select **Hidden**, **Bottom**, or **Right**.

To temporarily hide metrics for a graph on a dashboard

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. Select the color square for the metric that you want to hide in the graph's footer. An ***X*** appears in the color square when you hover over it, and the square turns gray when you choose it.
4. To restore the hidden metric, clear the **X** in the gray square.

Renaming a graph on a CloudWatch dashboard

You can change the default name that CloudWatch assigns to a graph on your dashboard.

To rename a graph on a dashboard

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. Hover over the title of the graph and choose **Widget actions** and **Edit**.
4. On the **Edit graph** screen, near the top, choose the title of the graph.
5. For **Title**, enter a new name and choose **Ok** (check mark). In the lower-right corner of the **Edit graph** screen, choose **Update widget**.

Moving a graph on a CloudWatch dashboard

To move a graph on a dashboard

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. Do one of the following:
   - Hover over the title of the graph until the selection icon appears. Select and drag the graph to a new location on the dashboard.
   - To move the widget to either the top left or bottom left of the dashboard, choose the vertical ellipsis at the upper right of the widget to open the **Widget actions** menu. Then choose **Move**, and choose where to move the widget to.
4. Choose **Save dashboard**.

Changing the size of graphs on CloudWatch dashboards

To change the size of a graph

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. To increase or decrease the size, hover over the graph and drag the lower right corner of the graph. Stop dragging the corner when you have the size that you want.
4. Choose **Save dashboard**.

Changing the size of a graph temporarily on a CloudWatch dashboard

To enlarge a graph temporarily

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. Select the graph. Alternatively, hover over the title of the graph and choose **Widget actions**, **Enlarge**.

Add a metrics explorer widget to a CloudWatch dashboard

The following steps explain how to add a metrics explorer widget to a dashboard using the console. You can also add it programmatically or by using Amazon CloudFormation. For more information, see Metrics Explorer Widget Object Definition and AWS::CloudWatch::Dashboard.

To add a metrics explorer widget to a dashboard

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Dashboards**.
3. Choose the name of the dashboard where you want to add the metrics explorer widget.
4. Choose the **+** symbol.
5. Choose **Explorer** and then choose **Next**.

   > **Note**
   >
   > You must be opted in to the new dashboard view to be able to add a Metrics Explorer widget. To opt in, choose **Dashboards** in the navigation pane, then choose **try out the new interface** in the banner at the top of the page.

6. Do one of the following:
   - To use a template, choose **Pre-filled Explorer widget** and then select a template to use.
   - To create a custom visualization, choose **Empty Explorer widget**.
7. Choose **Create**.

   If you used a template, the widget appears on your dashboard with the selected metrics. If you're satisfied with the explorer widget and the dashboard, choose **Save dashboard**.

   If you did not use a template, continue to the following steps.
8. In the new widget under **Explorer**, in the **Metrics** box, choose a single metric or all the available metrics from a service.

   After you choose a metric, you can optionally repeat this step to add more metrics.
9. For each metric selected, CloudWatch displays the statistic that it will use immediately after the metric name. To change this, choose the statistic name and then choose the statistic that you want.
10. Under **From**, choose a tag or a resource property to filter your results.

After you do this, you can optionally repeat this step to choose more tags or resource properties.

If you choose multiple values of the same property, such as two EC2 instance types, the explorer displays all the resources that match either chosen property. It's treated as an OR operation.

If you choose different properties or tags, such as the `Production` tag and the M5 instance type, only the resources that match all of these selections are displayed. This is treated as an AND operation.

11. (Optional) For **Aggregate by**, choose a statistic to use to aggregate the metrics. Then, next to **for**, choose how to aggregate the metric from the list. You can aggregate together all the resources that are currently displayed, or aggregate by a single tag or resource property.

    Depending on how you choose to aggregate, the result may be a single time series or multiple time series.
12. Under **Split by**, you can choose to split a single graph with multiple time series into multiple graphs. The split can be made by a variety of criteria, which you choose under **Split by**.
13. Under **Graph options**, you can refine the graph by changing the period, the type of graph, the legend placement, and the layout.
14. If you're satisfied with the explorer widget and the dashboard, choose **Save dashboard**.

Adding a line graph widget to a CloudWatch dashboard

To add a line widget to a dashboard

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. Choose the **+** symbol, and select **Line**.
4. Choose **Metrics**.
5. Choose **Browse**, and select the metric that you want to graph.
6. Choose **Create widget**, and then choose **Save dashboard**.

Removing a line graph widget from a CloudWatch dashboard

To remove a line graph widget from a dashboard

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. In the upper-right corner of the line widget that you want to remove, choose **Widget actions**, and then choose **Delete**.
4. Choose **Save dashboard**.

Adding a number widget to a CloudWatch dashboard

> **Note**
>
> Only the new interface supports the sparkline feature. When you create a number widget, the sparkline feature is automatically included.

To add a number widget to a dashboard

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. Choose the **+** symbol, and select **Number**.
4. In the **Browse** tab, search or browse for the metric that you want to display.
5. (Optional) To change the color of the sparkline feature, choose **Graphed metrics**, and select the color box next to the metric label. A menu appears where you can choose a different color or enter a six-digit hex color code to specify a color.
6. (Optional) To turn off the sparkline feature, choose **Options**. Under ***Sparkline***, the check box.
7. (Optional) To change your number widget's time range, select one of the predefined time ranges in the upper area of the widget. The time ranges span from 1 hour to 1 week (***1h***, ***3h***, ***12h***, ***1d***, ***3d***, or ***1w***).

   To set your own time range, choose **Custom**.

   - (Optional) To have this widget keep using this time range that you select, even if the time range for the rest of the dashboard is later changed, choose **Persist time range**.

8. (Optional) To have the number widget display an aggregate (***1h***, ***3h***, ***12h***, ***1d***, ***3d***, or ***1w***).

   To set your own time range, choose **Custom**.

   - (Optional) To have this widget display an average of the metric value over the entire time range, instead of the most recent value, choose **Options**, **Time range value shows the value from the entire time range**.

9. Choose **Create widget**, and choose **Save dashboard**.

> **Tip**
>
> You can turn off the sparkline feature from the number widget on the dashboard screen. In the upper-right corner of the number widget that you want to modify, choose **Widget actions**. Select **Sparkline**, and then choose **Hide sparkline**.

Removing a number widget from a CloudWatch dashboard

To remove a number widget from a dashboard

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose the dashboard that contains the number widget that you want to delete.
3. In the upper-right corner of the number widget that you want to remove, choose **Widget actions**, and then choose **Delete**.
4. Choose **Save dashboard**.

Adding a gauge widget to a CloudWatch dashboard

> **Note**
>
> Only the new interface in the CloudWatch console supports creation of the gauge widget. You must set a gauge range when you create this widget.

To add a gauge widget to a dashboard

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. From the dashboard screen, choose the **+** symbol, and then select **Gauge**.
4. Choose **Browse**, and then select the metric that you want to graph.
5. Choose **Options**. Under ***Gauge range***, set values for **Min** and **Max**. For percentages, such as CPU utilization, we recommend that you set the values for `Min` to `0` and `Max` to `100`.
6. (Optional) To change the color of the gauge widget, choose **Graphed metrics** and select the color box next to the metric label. A menu appears where you can choose a different color or enter a six-digit hex color code to specify a color.
7. Choose **Create widget**, and choose **Save dashboard**.

Removing a gauge widget from a CloudWatch dashboard

To remove a gauge widget from a dashboard

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose the dashboard that contains the gauge widget you want to delete.
3. In the upper-right corner of the gauge widget that you want to delete, choose **Widget actions**, and choose **Delete**.
4. Choose **Save dashboard**.

Using a custom widget on a CloudWatch dashboard

A *custom widget* is a CloudWatch dashboard widget that can call any Amazon Lambda function with custom parameters. It then displays the returned HTML or JSON. Custom widgets are a simple way to build a custom data view on a dashboard. If you can write Lambda code and create HTML, you can create a useful custom widget. Additionally, Amazon provides several prebuilt custom widgets that you can create without any code.

When you create a Lambda function to use as a custom widget, we strongly recommend that you include the prefix **customWidget** in the function name. This helps you know which of your Lambda functions are safe to use when you add custom widgets to your dashboard.

Custom widgets behave like other widgets on your dashboard. They can be refreshed and auto-refreshed, resized, and moved around. They react to the time range of the dashboard.

If you have set up CloudWatch console cross-account functionality, you can add a custom widget created in one account to dashboards in other accounts. For more information, see Cross-account cross-Region CloudWatch console.

You can also use custom widgets on your own website by using the CloudWatch dashboard sharing feature. For more information, see Sharing CloudWatch dashboards.

Topics

- Details about custom widgets in CloudWatch
- Security and JavaScript for custom CloudWatch widgets
- Interactivity in the custom widget in CloudWatch
- Creating a custom widget for a CloudWatch dashboard
- Sample custom widgets for a CloudWatch dashboard

Details about custom widgets in CloudWatch

Custom widgets work as follows:

1. The CloudWatch dashboard calls the Lambda function containing the widget code. It passes in any custom parameters that are defined in the widget.
2. The Lambda function returns a string of HTML, JSON, or Markdown. Markdown is returned as JSON in the following format:

   ```
   {"markdown":"markdown content"}
   ```

3. The dashboard displays the returned HTML or JSON.

If the function returns HTML, most HTML tags are supported. You can use Cascading Style Sheets (CSS) styles and Scalable Vector Graphics (SVG) to build sophisticated views.

The default style of HTML elements such as links and tables follow the styling of CloudWatch dashboards. You can customize this style by using inline styles, using the `<style>` tag. You can also deactivate the default styles by including a single HTML element with the class of `cwdb-no-`

`default-styles`. The following example deactivates default styles: `<div class="cwdb-no-default-styles"></div>`.

Every call by a custom widget to Lambda includes a `widgetContext` element with the following contents, to provide the Lambda function developer with useful context information.

```
{
  "widgetContext": {
    "dashboardName": "Name-of-current-dashboard",
    "widgetId": "widget-16",
    "accountId": "012345678901",
    "locale": "en",
    "timezone": {
      "label": "UTC",
      "offsetISO": "+00:00",
      "offsetInMinutes": 0
    },
    "period": 300,
    "isAutoPeriod": true,
    "timeRange": {
      "mode": "relative",
      "start": 1627236199729,
      "end": 1627322599729,
      "relativeStart": 86400012,
      "zoom": {
        "start": 1627276030434,
        "end": 1627282956521
      }
    },
    "theme": "light",
    "linkCharts": true,
    "title": "Tweets for Amazon website problem",
    "forms": {
      "all": {}
    },
    "params": {
      "original": "param-to-widget"
    },
    "width": 588,
    "height": 369
  }
}
```

Default CSS styling

Custom widgets provide the following default CSS styling elements:

- You can use the CSS class **btn** to add a button. It turns an anchor (<a>) into a button as in the following example:

```
<a class="btn" href=https://amazon.com">Open Amazon</a>
```

- You can use the CSS class **btn btn-primary** to add a primary button.
- The following elements are styled by default: **table**, **select**, **headers (h1, h2, and h3)**, **preformatted text (pre)**, **input**, and **text area**.

Using the describe parameter

We strongly recommend that you support the **describe** parameter in your functions, even if it just returns an empty string. If you don't support it, and it is called in the custom widget, it displays widget content as if it was documentation.

If you include the **describe** parameter, the Lambda function returns the documentation in Markdown format and does nothing else.

When you create a custom widget in the console, after you select the Lambda function a **Get documentation** button appears. If you choose this button, the function is invoked with the **describe** parameter and the function's documentation is returned. If the documentation is well-formatted in markdown, CloudWatch parses the first entry in the documentation that is surrounded by three single backtick characters (```) in YAML. Then, it automatically populates the documentation in the parameters. The following is an example of such well-formatted documentation.

````
``` yaml
echo: <h1>Hello world</h1>
```
````

Security and JavaScript for custom CloudWatch widgets

For security reasons, JavaScript is not allowed in the returned HTML. Removing the JavaScript prevents permission escalation issues, where the writer of the Lambda function injects code that could run with higher permissions than the user viewing the widget on the dashboard.

If the returned HTML contains any JavaScript code or other known security vulnerabilities, it is cleaned from the HTML before it is rendered on the dashboard. For example, the **<iframe>** and **<use>** tags are not allowed and are removed.

Custom Widgets won't run by default in a dashboard. Instead, you must explicitly allow a custom widget to run if you trust the Lambda function that it invokes. You can choose to allow it once or allow always, for both individual widgets and entire dashboard. You can also deny permission for individual widgets and the entire dashboard.

Interactivity in the custom widget in CloudWatch

Even though JavaScript is not allowed, there are other ways to allow interactivity with the returned HTML.

- Any element in the returned HTML can be tagged with special configuration in a `<cwdb-action>` tag, which can display information in pop-ups, ask for confirmation on clicks, and call any Lambda function when that element is chosen. For example, you can define buttons that call any Amazon API using a Lambda function. The returned HTML can be set to either replace the existing Lambda widget's content, or display inside a modal.
- The returned HTML can include links that open new consoles, open other customer pages, or load other dashboards.
- The HTML can include the `title` attribute for an element, which gives additional information if the user hovers over that element.
- The element can include CSS selectors, such as `:hover`, which can invoke animations or other CSS effects. You can also show or hide elements in the page.

<cwdb-action> definition and usage

The `<cwdb-action>` element defines a behavior on the immediately previous element. The content of the `<cwdb-action>` is either HTML to display or a JSON block of parameters to pass to a Lambda function.

The following is an example of a `<cwdb-action>` element.

```
<cwdb-action
     action="call|html"
     confirmation="message"
     display="popup|widget"
     endpoint="<lambda ARN>"
     event="click|dblclick|mouseenter">

     html | params in JSON
</cwdb-action>
```

- **action**— Valid values are `call`, which calls a Lambda function, and `html`, which displays any HTML contained within `<cwdb-action>`. The default is `html`.
- **confirmation**— Displays a confirmation message that must be acknowledged before the action is taken, allowing the customer to cancel.
- **display**— Valid values are `popup` and `widget`, which replaces the content of the widget itself. The default is `widget`.
- **endpoint**— The Amazon Resource Name (ARN) of the Lambda function to call. This is required if `action` is `call`.
- **event**— Defines the event on the previous element that invokes the action. Valid values are `click`, `dblclick`, and `mouseenter`. The `mouseenter` event can be used only in combination with the `html` action. The default is `click`.

Examples

The following is an example of how to use the `<cwdb-action>` tag to create a button that reboots an Amazon EC2 instance using a Lambda function call. It displays the success or failure of the call in a pop-up.

```
<a class="btn">Reboot Instance</a>
<cwdb-action action="call" endpoint="arn:aws:lambda:us-
east-1:123456:function:rebootInstance" display="popup">
       { "instanceId": "i-342389adbfef" }
</cwdb-action>
```

The next example displays more information in a pop-up.

```
<a>Click me for more info in popup</a>
<cwdb-action display="popup">
```

```
    <h1>Big title</h1>
    More info about <b>something important</b>.
</cwdb-action>
```

This example is a **Next** button that replaces the content of a widget with a call to a Lambda function.

```
<a class="btn btn-primary">Next</a>
<cwdb-action action="call" endpoint="arn:aws:lambda:us-
east-1:123456:function:nextPage">
    { "pageNum": 2 }
</cwdb-action>
```

Creating a custom widget for a CloudWatch dashboard

To create a custom widget, you can use one of the samples provided by Amazon, or you can create your own. The Amazon samples include samples in both JavaScript and Python, and are created by a Amazon CloudFormation stack. For a list of samples, see Sample custom widgets for a CloudWatch dashboard.

To create a custom widget in a CloudWatch dashboard

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. Choose the **+** symbol.
4. Choose **Custom widget**.
5. Use one of the following methods:
   - To use a sample custom widget provided by Amazon, do the following:
     a. Select the sample in the dropdown box.

        The Amazon CloudFormation console launches in a new browser. In the Amazon CloudFormation console, do the following:
     b. (Optional) Customize the Amazon CloudFormation stack name.
     c. Make selections for any parameters used by the sample.
     d. Select **I acknowledge that Amazon CloudFormation might create IAM resources**, and choose **Create stack**.
   - To create your own custom widget provided by Amazon, do the following:

a. Choose **Next**.

b. Choose to either select your Lambda function from a list, or enter its Amazon Resource Name (ARN). If you select it from a list, also specify the Region where the function is and the version to use.

c. For **Parameters**, make selections for any parameters used by the function.

d. Enter a title for the widget.

e. For **Update on**, configure when the widget should be updated (when the Lambda function should be called again). This can be one or more of the following: **Refresh** to update it when the dashboard auto-refreshes, **Resize** to update it whenever the widget is resized, or **Time Range** to update it whenever the dashboard's time range is adjusted, including when graphs are zoomed into.

f. If you are satisfied with the preview, choose **Create widget**.

Sample custom widgets for a CloudWatch dashboard

Amazon provides sample custom widgets in both JavaScript and Python. You can create these sample widgets by using the link for each widget in this list. Alternatively, you can create and customize a widget by using the CloudWatch console. The links in this list open an Amazon CloudFormation console and use an Amazon CloudFormation quick-create link to create the custom widget.

You can also access the custom widget samples on GitHub.

Following this list, complete examples of the Echo widget are shown for each language.

JavaScript

Sample custom widgets in JavaScript

- Echo – A basic echoer that you can use to test how HTML appears in a custom widget, without having to write a new widget.
- Hello world – A very basic starter widget.
- Custom widget debugger – A debugger widget that displays useful information about the Lambda runtime environment.
- Query CloudWatch Logs Insights – Run and edit CloudWatch Logs Insights queries.
- Run Amazon Athena queries – Run and edit Athena queries.

- Call Amazon API – Call any read-only Amazon API and display the results in JSON format.
- Fast CloudWatch bitmap graph – Render CloudWatch graphs using on the server side, for fast display.
- Text widget from CloudWatch dashboard – Displays the first text widget from the specified CloudWatch dashboard.
- CloudWatch metric data as a table – Displays raw CloudWatch metric data in a table.
- Amazon EC2 table – Displays the top EC2 instances by CPU utilization. This widget also includes a Reboot button, which is disabled by default.
- Amazon CodeDeploy deployments – Displays CodeDeploy deployments.
- Amazon Cost Explorer report – Displays a report on the cost of each Amazon service for a selected time range.
- Display content of external URL – Displays the content of an externally accessible URL.
- Display an Amazon S3 object – Displays an object in an Amazon S3 bucket in your account.
- Simple SVG pie chart – Example of a graphical SVG-based widget.

Python

Sample custom widgets in Python

- Echo – A basic echoer which you can use to test how HTML appears in a custom widget, without having to write a new widget.
- Hello world – A very basic starter widget.
- Custom widget debugger – A debugger widget that displays useful information about the Lambda runtime environment.
- Call Amazon API – Call any read-only Amazon API and display the results in JSON format.
- Fast CloudWatch bitmap graph – Render CloudWatch graphs using on the server side, for fast display.
- Send dashboard snapshot by email – Take a snapshot of the current dashboard and send it to email recipients.
- Send dashboard snapshot to Amazon S3 – Take a snapshot of the current dashboard and store it in Amazon S3.
- Text widget from CloudWatch dashboard – Displays the first text widget from the specified CloudWatch dashboard.
- Display content of external URL – Displays the content of an externally accessible URL.

- RSS reader – Displays RSS feeds.
- Display an Amazon S3 object – Displays an object in an Amazon S3 bucket in your account.
- Simple SVG pie chart – Example of a graphical SVG-based widget.

Echo widget in JavaScript

The following is the Echo sample widget in JavaScript.

```
const DOCS = `
## Echo
A basic echo script. Anything passed in the \`\`\`echo\`\`\` parameter is returned as
 the content of the custom widget.
### Widget parameters
Param | Description
---|---
**echo** | The content to echo back

### Example parameters
\`\`\` yaml
echo: <h1>Hello world</h1>
\`\`\`
`;

exports.handler = async (event) => {
    if (event.describe) {
        return DOCS;
    }

    let widgetContext = JSON.stringify(event.widgetContext, null, 4);
    widgetContext = widgetContext.replace(/</g, '&lt;');
    widgetContext = widgetContext.replace(/>/g, '&gt;');

    return `${event.echo || ''}<pre>${widgetContext}</pre>`;
};
```

Echo widget in Python

The following is the Echo sample widget in Python.

```
import json

DOCS = """
```

```
## Echo
A basic echo script. Anything passed in the ```echo``` parameter is returned as the
 content of the custom widget.
### Widget parameters
Param | Description
---|---
**echo** | The content to echo back

### Example parameters
``` yaml
echo: <h1>Hello world</h1>
```"""

def lambda_handler(event, context):
    if 'describe' in event:
        return DOCS

    echo = event.get('echo', '')
    widgetContext = event.get('widgetContext')
    widgetContext = json.dumps(widgetContext, indent=4)
    widgetContext = widgetContext.replace('<', '<')
    widgetContext = widgetContext.replace('>', '>')

    return f'{echo}<pre>{widgetContext}</pre>'
```

Echo widget in Java

The following is the Echo sample widget in Java.

```
package example;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;

import com.google.gson.Gson;
import com.google.gson.GsonBuilder;

public class Handler implements RequestHandler<Event, String>{

  static String DOCS = ""
    + "## Echo\n"
    + "A basic echo script. Anything passed in the ```echo``` parameter is returned as
 the content of the custom widget.\n"
    + "### Widget parameters\n"
```

```
    + "Param | Description\n"
    + "---|---\n"
    + "**echo** | The content to echo back\n\n"
    + "### Example parameters\n"
    + "```yaml\n"
    + "echo: <h1>Hello world</h1>\n"
    + "```\n";

  Gson gson = new GsonBuilder().setPrettyPrinting().create();

  @Override
  public String handleRequest(Event event, Context context) {

    if (event.describe) {
      return DOCS;
    }

    return (event.echo != null ? event.echo : "") + "<pre>" +
 gson.toJson(event.widgetContext) + "</pre>";
  }
}

class Event {

    public boolean describe;
    public String echo;
    public Object widgetContext;

    public Event() {}

    public Event(String echo, boolean describe, Object widgetContext) {
        this.describe = describe;
        this.echo = echo;
        this.widgetContext = widgetContext;
    }
}
```

Adding a text widget to a CloudWatch dashboard

To add a text widget to a dashboard

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.

3. Choose the **+** symbol.
4. Choose **Text**.
5. For **Markdown**, add and format your text using Markdown and choose **Create widget**.
6. To make the text widget transparent, choose **Transparent background**.
7. Choose **Save dashboard**.

Editing a text widget on a CloudWatch dashboard

To edit a text widget on a dashboard

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. Hover over the upper-right corner of the text block and choose **Widget actions**. Then, choose **Edit**.
4. Update the text as needed and choose **Update widget**.
5. Choose **Save dashboard**.

Removing a text widget from a CloudWatch dashboard

To remove a text widget from a dashboard

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. Hover over the upper-right corner of the text block and choose **Widget actions**. Then, choose **Delete**.
4. Choose **Save dashboard**.

Adding an alarm to a CloudWatch dashboard

To add a single alarm, including its graph, to a dashboard

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Alarms**, select the alarm to add, and then choose **Add to Dashboard**.

3. Select a dashboard, choose a widget type (**Line**, **Stacked area**, or **Number**), and then choose **Add to dashboard**.
4. To see your alarm on the dashboard, choose **Dashboards** in the navigation pane and select the dashboard.
5. (Optional) To temporarily make an alarm graph larger, select the graph.
6. (Optional) To change the widget type, hover over the title of the graph, choose **Widget actions**, and then choose **Widget type**.

Adding an alarm status widget to a CloudWatch dashboard

To add an alarm status widget to a dashboard

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. Choose the **+** symbol.
4. Choose **Alarm status**.
5. Select the check boxes next to the alarms that you want to add to the widget, and then choose **Create widget**.
6. Choose **Add to dashboard**.

Removing an alarm widget from a CloudWatch dashboard

To remove an alarm widget from a dashboard

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. Hover over the widget, choose **Widget actions**, and then choose **Delete**.
4. Choose **Save dashboard**. If you attempt to navigate away from the dashboard before you save your changes, you're prompted to either save or discard your changes.

Using a data table widget in a CloudWatch dashboard

Table properties

A data table has a default set of properties that don't require any changes to be made to the options or source. These properties include a sticky label column, all summary columns enabled, datapoints rounded, and their units converted.

Each data table widget can have the following properties. The information about each property includes how to configure it in the JSON source of the dashboard. For more information about dashboard JSON, see [Dashboard Body Structure and Syntax](#).

Summary

Summary columns are a new property introduced with the data table widget. These columns are a specific subset of summaries of your current table. For example, the **Sum** summary is a sum of all displayed datapoints in its row. The summary columns are not the same as CloudWatch statistics. Represented in source as:

```
"table": {
       "summaryColumns": [
           "MIN",
           "MAX",
           "SUM",
           "AVG"
       ]
   },
```

Thresholds

Use this to apply thresholds to your table. When a data point falls within a threshold, its cell is highlighted with the threshold color. Represented in source as:

```
"annotations": {
    "horizontal": [
        {
            "label": string,
            "value": int,
            "fill": "above" | "below"
        }
    ]
}
```

Unit in label column

To display what unit is associated with the metric, you can enable this option to display the unit in the label column beside the label. Represented in source as:

```
"yAxis": {
    "left": {
        "showUnits": true | false
    }
}
```

Invert rows and columns

This transforms the table so that the datapoints swap from columns to rows, and the metrics become columns. Represented in source as:

```
 "table": {
    "layout": "vertical" | "horizontal"
}
```

Sticky summary columns

This makes the summary columns sticky, so that they remain in view while you scroll. The label is already sticky. Represented in source as:

```
"table": {
    "stickySummary": true | false
}
```

Display only summary columns

This prevents the columns of datapoints from being displayed, so that only the label and summary columns are displayed. Represented in source as:

```
 "table": {
    "showTimeSeriesData": false | true
}
```

Live data

Displays the most recent data point, even if it is not yet fully aggregated. Represented in source as:

```
"liveData": true | false
```

Number widget format

Displays as many digits as can fit in the cell, before rounding and converting. Represented in source as:

```
"singleValueFullPrecision": true | false
```

Adding a data table widget to a CloudWatch dashboard

To add a data table widget to a dashboard

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Dashboards** and then choose a dashboard.
3. Choose the **+** button, select **Data table**, and choose **Next**.
4. In the **Browse** tab, search or browse for the metrics that you want to display in the table widget. Then select the metrics.
5. (Optional) To change the layout of the table, choose the **Options** tab and select **Invert rows and columns**.

   You can also use the **Options** tab to change what columns appear in the table and display the unit being used in the **Label** column.

   > **Tip**
   >
   > To display more accurate thresholds, choose **Show as many digits as can fit before rounding**.

6. (Optional) To change your data table widget's time range, select one of the predefined time ranges in the upper area of the widget. The time ranges span from 1 hour to 1 week. To set your own time range, choose **Custom**.
7. (Optional) To change your data table widget's time range, select one of the predefined time ranges in the upper area of the widget. The time ranges span from 1 hour to 1 week. To set your own time range, choose **Custom**.

8. (Optional) To have this widget keep using the time range that you select, even if the time range for the rest of the dashboard is later changed, choose **Persist time range**.
9. Choose **Create widget** and then choose **Save dashboard**.

Removing a data table widget from a CloudWatch dashboard

To remove a data table widget to a dashboard

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. In the upper-right corner of the widget that you want to remove, choose **Widget actions**, **Delete**.
4. Choose **Save dashboard**.

Linking graphs on a CloudWatch dashboard

Link graphs on a CloudWatch dashboard so that changes in time are reflected on all graphs.

To link the graphs on a dashboard

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. Choose **Actions** and then **Link graphs**.

Unlinking graphs on a CloudWatch dashboard

To unlink the graphs on a dashboard

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. Clear **Actions** and then **Link graphs**.

Sharing CloudWatch dashboards

You can share your CloudWatch dashboards with people who do not have direct access to your Amazon account. This enables you to share dashboards across teams, with stakeholders, and with people external to your organization. You can even display dashboards on big screens in team areas, or embed them in Wikis and other webpages.

> **Warning**
>
> All people who you share the dashboard with are granted the permissions listed in Permissions that are granted to people who you share the dashboard with for the account. If you share the dashboard publicly, then everyone who has the link to the dashboard has these permissions.
> The `cloudwatch:GetMetricData` and `ec2:DescribeTags` permissions cannot be scoped down to specific metrics or EC2 instances, so the people with access to the dashboard can query all CloudWatch metrics and the names and tags of all EC2 instances in the account.

When you share dashboards, you can designate who can view the dashboard in three ways:

- Share a single dashboard and designate as many as five email addresses of people who can view the dashboard. Each of these users creates their own password that they must enter to view the dashboard.
- Share a single dashboard publicly, so that anyone who has the link can view the dashboard.
- Share all the CloudWatch dashboards in your account and specify a third-party single sign-on (SSO) provider for dashboard access. All users who are members of this SSO provider's list can access all the dashboards in the account. To enable this, you integrate the SSO provider with Amazon Cognito. The SSO provider must support Security Assertion Markup Language (SAML). For more information about Amazon Cognito, see What is Amazon Cognito?

Sharing a dashboard doesn't incur charges, but widgets inside a shared dashboard incur charges at standard CloudWatch rates. For more information about CloudWatch pricing, see Amazon CloudWatch Pricing.

When you share a dashboard, Amazon Cognito resources are created in the US East (N. Virginia) Region.

Important

Do not modify resource names and identifiers that are created by the dashboard sharing process. This includes Amazon Cognito and IAM resources. Modifying these resources can cause unexpected and incorrect functionality of shared dashboards.

Note

If you share a dashboard that has metric widgets with alarm annotations, the people that you share the dashboard with will not see those widgets. They will instead see a blank widget with text saying that the widget is not available. You will still see metric widgets with alarm annotations when you view the dashboard yourself.

Permissions required to share a dashboard

To be able to share dashboards using any of the following methods and to see which dashboards have already been shared, you must be signed on as a user or with an IAM role that has certain permissions.

To be able to share dashboards, your user or IAM role must include the permissions included in the following policy statement:

```
{
    "Effect": "Allow",
    "Action": [
        "iam:CreateRole",
        "iam:CreatePolicy",
        "iam:AttachRolePolicy",
        "iam:PassRole"
    ],
    "Resource": [
        "arn:aws:iam::*:role/service-role/CWDBSharing*",
        "arn:aws:iam::*:policy/*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
```

```
            "cognito-idp:*",
            "cognito-identity:*",
        ],
        "Resource": [
            "*"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "cloudwatch:GetDashboard",
        ],
        "Resource": [
            "*"
            // or the ARNs of dashboards that you want to share
        ]
    }
```

To be able to see which dashboards are shared, but not be able to share dashboards, a user or an IAM role can include a policy statement similar to the following:

```
{
    "Effect": "Allow",
    "Action": [
        "cognito-idp:*",
        "cognito-identity:*"
    ],
    "Resource": [
        "*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "cloudwatch:ListDashboards",
    ],
    "Resource": [
        "*"
    ]
}
```

Permissions that are granted to people who you share the dashboard with

When you share a dashboard, CloudWatch creates an IAM role in the account which gives the following permissions to the people who you share the dashboard with:

- `cloudwatch:GetInsightRuleReport`
- `cloudwatch:GetMetricData`
- `cloudwatch:DescribeAlarms`
- `ec2:DescribeTags`

> **Warning**
>
> All people who you share the dashboard with are granted these permissions for the account. If you share the dashboard publicly, then everyone who has the link to the dashboard has these permissions.
> The `cloudwatch:GetMetricData` and `ec2:DescribeTags` permissions cannot be scoped down to specific metrics or EC2 instances, so the people with access to the dashboard can query all CloudWatch metrics and the names and tags of all EC2 instances in the account.

When you share a dashboard, by default the permissions that CloudWatch creates restrict access to only the alarms and Contributor Insights rules that are on the dashboard when it is shared. If you add new alarms or Contributor Insights rules to the dashboard and want them to also be seen by the people who you shared the dashboard with, you must update the policy to allow these resources.

Sharing a CloudWatch dashboard with specific users

Use the steps in this section to share a dashboard with as many as five email addresses that you choose.

> **Note**
>
> By default, any CloudWatch Logs widgets on the dashboard are not visible to people who you share the dashboard with. For more information, see Allowing people that you share with to see logs table widgets.
>
> By default, any composite alarm widgets on the dashboard are not visible to people who you share the dashboard with. For more information, see Allowing people that you share with to see composite alarms.

To share a dashboard with specific users

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Dashboards**.
3. Choose the name of your dashboard.
4. Choose **Actions**, **Share dashboard**.
5. Next to **Share your dashboard and require a username and password**, choose **Start sharing**.
6. Under **Add email addresses**, enter the email addresses that you want to share the dashboard with. You can include as many as five email addresses.
7. When you have all the email addresses entered, read the agreement and select the confirmation box. Then choose **Preview policy**.
8. Confirm that the resources that will be shared are what you want, and choose **Confirm and generate shareable link**.
9. On the next page, choose **Copy link to clipboard**. You can then paste this link into email and send it to the invited users. They automatically receive a separate email with their user name and a temporary password to use to connect to the dashboard.

Sharing a CloudWatch dashboard publicly

Follow the steps in this section to share a dashboard publicly. This can be useful to display the dashboard on a big screen in a team room, or embed it in a Wiki page.

> **Important**
>
> Sharing a dashboard publicly makes it accessible to anyone who has the link, with no authentication. Do this only for dashboards that do not contain sensitive information.

> **Note**
>
> By default, any CloudWatch Logs widgets on the dashboard are not visible to people who you share the dashboard with. For more information, see Allowing people that you share with to see logs table widgets.
> By default, any composite alarm widgets on the dashboard are not visible to people who you share the dashboard with. For more information, see Allowing people that you share with to see composite alarms.

To share a dashboard publicly

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Dashboards**.
3. Choose the name of your dashboard.
4. Choose **Actions**, **Share dashboard**.
5. Next to **Share your dashboard publicly**, choose **Start sharing**.
6. Enter **`Confirm`** in the text box.
7. Read the agreement and select the confirmation box. Then choose **Preview policy**.
8. Confirm that the resources that will be shared are what you want, and choose **Confirm and generate shareable link**.
9. On the next page, choose **Copy link to clipboard**. You can then share this link. Anyone you share the link with can access the dashboard, without providing credentials.

Sharing all CloudWatch dashboards in the account by using SSO

Use the steps in this section to share all the dashboards in your account with users by using single sign-on (SSO).

Note

By default, any CloudWatch Logs widgets on the dashboard are not visible to people who you share the dashboard with. For more information, see Allowing people that you share with to see logs table widgets.

By default, any composite alarm widgets on the dashboard are not visible to people who you share the dashboard with. For more information, see Allowing people that you share with to see composite alarms.

To share your CloudWatch dashboards with users who are in an SSO provider's list

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Dashboards**.
3. Choose the name of your dashboard.
4. Choose **Actions**, **Share dashboard**.
5. Choose **Go to CloudWatch Settings**.
6. If the SSO provider that you want isn't listed in **Available SSO providers**, choose **Manage SSO providers** and follow the instructions in Setting up SSO for CloudWatch dashboard sharing.

   Then return to the CloudWatch console and refresh the browser. The SSO provider that you enabled should now appear in the list.
7. Choose the SSO provider that you want in the **Available SSO providers** list.
8. Choose **Save changes**.

Setting up SSO for CloudWatch dashboard sharing

To set up dashboard sharing through a third-party single sign-on provider that supports SAML, follow these steps.

Important

We strongly recommend that you do not share dashboards using a non-SAML SSO provider. Doing so causes a risk of inadvertently allowing third parties to access your account's dashboards.

To set up an SSO provider to enable dashboard sharing

1. Integrate the SSO provider with Amazon Cognito. For more information, see [Integrating Third-Party SAML Identity Providers with Amazon Cognito User Pools](#).
2. Download the metadata XML file from your SSO provider.
3. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
4. In the navigation pane, choose **Settings**.
5. In the **Dashboard sharing** section, choose **Configure**.
6. Choose **Manage SSO providers**.

   This opens the Amazon Cognito console in the US East (N. Virginia) Region (us-east-1). If you don't see any **User Pools**, the Amazon Cognito console might have opened in a different Region. If so, change the Region to **US East (N. Virginia) us-east-1** and proceed with the next steps.
7. Choose the **CloudWatchDashboardSharing** pool.
8. In the navigation pane, choose **Social and external providers**.
9. Choose **Add identity provider**.
10. Choose **SAML**.
11. Enter a name for your SSO provider in **Provider name**.
12. Choose **Select file**, and select the metadata XML file that you downloaded in step 2.
13. Choose **Create provider**.

Seeing how many of your CloudWatch dashboards are shared

You can use the CloudWatch console to see how many of your CloudWatch dashboards are currently being shared with others.

To see how many of your dashboards are being shared

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Settings**.
3. The **Dashboard sharing** section displays how many dashboards are shared.
4. To see which dashboards are shared, choose ***number*** **dashboards shared** under **Username and password** and under **Public dashboards.**

Seeing which of your CloudWatch dashboards are shared

You can use the CloudWatch console to see which of your dashboards are currently being shared with others.

To see which of your dashboards are being shared

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Dashboards**.
3. In the list of dashboards, see the **Share** column. Dashboards that have the icon in this column filled in are currently being shared.
4. To see which users a dashboard is being shared with, choose the dashboard name, and then choose **Actions**, **Share dashboard**.

   The **Share dashboard** ***`dashboard name`*** page displays how the dashboard is being shared. If you want, you can stop sharing the dashboard by choosing **Stop sharing**.

Stopping sharing one or more CloudWatch dashboards

You can stop sharing a single shared dashboard, or stop sharing all shared dashboards at once.

To stop sharing a single dashboard

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Dashboards**.
3. Choose the name of the shared dashboard.
4. Choose **Actions**, **Share dashboard**.
5. Choose **Stop sharing**.
6. In the confirmation box, choose **Stop sharing**.

To stop sharing all shared dashboards

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Settings**.
3. In the **Dashboard sharing** section, choose **Stop sharing all dashboards**.
4. In the confirmation box, choose **Stop sharing all dashboards**.

Reviewing shared CloudWatch dashboard permissions and changing permission scope

Use the steps in this section if you want to review the permissions of the users of your shared dashboards, or change the scope of shared dashboard permissions.

To review shared dashboard permissions

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Dashboards**.
3. Choose the name of the shared dashboard.
4. Choose **Actions**, **Share dashboard**.
5. Under **Resources**, choose **IAM Role**.
6. In the IAM console, choose the displayed policy.
7. (Optional) To limit which alarms that shared dashboard users can see, choose **Edit policy** and move the `cloudwatch:DescribeAlarms` permission from its current position to a new `Allow` statement that lists the ARNs of only the alarms that you want to be seen by shared dashboard users. See the following example.

```
{
   "Effect": "Allow",
    "Action": "cloudwatch:DescribeAlarms",
    "Resource": [
        "AlarmARN1",
        "AlarmARN2"
    ]
}
```

   If you do this, be sure to remove the `cloudwatch:DescribeAlarms` permission from a section of the current policy that looks like this:

```
{
   "Effect": "Allow",
    "Action": [
        "cloudwatch:GetInsightRuleReport",
        "cloudwatch:GetMetricData",
        "cloudwatch:DescribeAlarms",
        "ec2:DescribeTags"
```

```
    ],
    "Resource": "*"
}
```

8. (Optional) To limit the scope of what Contributor Insights rules that shared dashboard users can see, choose **Edit policy** and move the `cloudwatch:GetInsightRuleReport` from its current position to a new `Allow` statement that lists the ARNs of only the Contributor Insights rules that you want to be seen by shared dashboard users. See the following example.

```
{
   "Effect": "Allow",
    "Action": "cloudwatch:GetInsightRuleReport",
    "Resource": [
        "PublicContributorInsightsRuleARN1",
        "PublicContributorInsightsRuleARN2"
    ]
}
```

   If you do this, be sure to remove `cloudwatch:GetInsightRuleReport` from a section of the current policy that looks like this:

```
{
            "Effect": "Allow",
            "Action": [
                "cloudwatch:GetInsightRuleReport",
                "cloudwatch:GetMetricData",
                "cloudwatch:DescribeAlarms",
                "ec2:DescribeTags"
            ],
            "Resource": "*"
        }
```

Allowing people that you share with to see composite alarms

When you share a dashboard, by default the composite alarm widgets on the dashboard are not visible to the people who you share the dashboard with. For composite alarm widgets to be visible, you need to add a `DescribeAlarms: *` permission to the dashboard sharing policy. That permission would look like this:

```
{
```

```
    "Effect": "Allow",
    "Action": "cloudwatch:DescribeAlarms",
    "Resource": "*"
}
```

> **Warning**
>
> The preceding policy statement give access to all alarms in the account. To reduce the scope of `cloudwatch:DescribeAlarms`, you must use a `Deny` statement. You can add a `Deny` statement to the policy and specify the ARNs of the alarms that you want to lock down. That deny statement should look similar to the following:
>
> ```
> {
>     "Effect": "Allow",
>     "Action": "cloudwatch:DescribeAlarms",
>     "Resource": "*"
> },
> {
>     "Effect": "Deny",
>     "Action": "cloudwatch:DescribeAlarms",
>     "Resource": [
>         "SensitiveAlarm1ARN",
>         "SensitiveAlarm1ARN"
>     ]
> }
> ```

Allowing people that you share with to see logs table widgets

When you share a dashboard, by default the CloudWatch Logs Insights widgets that are on the dashboard are not visible to the people who you share the dashboard with. This affects both CloudWatch Logs Insights widgets that exist now and any that are added to the dashboard after you share it.

If you want these people to be able to see CloudWatch Logs widgets, you must add permissions to the IAM role for dashboard sharing.

To allow the people that you share a dashboard with to see the CloudWatch Logs widgets

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.

2. In the navigation pane, choose **Dashboards**.
3. Choose the name of the shared dashboard.
4. Choose **Actions**, **Share dashboard**.
5. Under **Resources**, choose **IAM Role**.
6. In the IAM console, choose the displayed policy.
7. Choose **Edit policy** and add the following statement. In the new statement, we recommend that you specify the ARNs of only the log groups that you want shared. See the following example.

```
{
            "Effect": "Allow",
            "Action": [
                "logs:FilterLogEvents",
                "logs:StartQuery",
                "logs:StopQuery",
                "logs:GetLogRecord",
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "SharedLogGroup1ARN",
                "SharedLogGroup2ARN"
           ]
        },
```

8. Choose **Save Changes**.

If your IAM policy for dashboard sharing already includes those five permissions with * as the resource, we strongly recommend that you change the policy and specify only the ARNs of the log groups that you want shared. For example, if your `Resource` section for these permissions was the following:

```
"Resource": "*"
```

Change the policy to specify only the ARNs of the log groups that you want shared, as in the following example:

```
"Resource": [
  "SharedLogGroup1ARN",
  "SharedLogGroup2ARN"
```

```
]
```

Allowing people that you share with to see custom widgets

When you share a dashboard, by default the custom widgets that are on the dashboard are not visible to the people who you share the dashboard with. This affects both custom widgets that exist now and any that are added to the dashboard after you share it.

If you want these people to be able to see custom widgets, you must add permissions to the IAM role for dashboard sharing.

To allow the people that you share a dashboard with to see the custom widgets

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Dashboards**.
3. Choose the name of the shared dashboard.
4. Choose **Actions**, **Share dashboard**.
5. Under **Resources**, choose **IAM Role**.
6. In the IAM console, choose the displayed policy.
7. Choose **Edit policy** and add the following statement. In the new statement, we recommend that you specify the ARNs of only the Lambda functions that you want shared. See the following example.

```
{
  "Sid": "Invoke",
  "Effect": "Allow",
  "Action": [
      "lambda:InvokeFunction"
  ],
  "Resource": [
    "LambdaFunction1ARN",
    "LambdaFunction2ARN"
  ]
 }
```

8. Choose **Save Changes**.

If your IAM policy for dashboard sharing already includes that permission with * as the resource, we strongly recommend that you change the policy and specify only the ARNs of the Lambda functions that you want shared. For example, if your `Resource` section for these permissions was the following:

```
"Resource": "*"
```

Change the policy to specify only the ARNs of the custom widgets that you want shared, as in the following example:

```
"Resource": [
  "LambdaFunction1ARN",
  "LambdaFunction2ARN"
]
```

Using live data in CloudWatch dashboards

You can choose whether your metric widgets display *live data*. Live data is data published within the last minute that has not been fully aggregated.

- If live data is turned **off**, only data points with an aggregation period of at least one minute in the past are shown. For example, when using 5-minute periods, the data point for 12:35 would be aggregated from 12:35 to 12:40, and displayed at 12:41.
- If live data is turned **on**, the most recent data point is shown as soon as any data is published in the corresponding aggregation interval. Each time you refresh the display, the most recent data point may change as new data within that aggregation period is published. If you use a cumulative statistic such as **Sum** or **Sample Count**, using live data may result in a dip at the end of your graph.

You can choose to enable live data for a whole dashboard, or for individual widgets on the dashboard.

Using live data for a CloudWatch dashboard

To choose whether to use live data on your entire dashboard

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. To permanently turn live data on or off for all widgets on the dashboard, do the following:

   a. Choose **Actions**, **Settings**, **Bulk update live data.**
   b. Choose **Live Data on** or **Live Data off**, and choose **Set**.

4. To temporarily override the live data settings of each widget, choose **Actions**. Then, under **Overrides**, next to **Live data**, do one of the following:

   - Choose **On** to temporarily turn on live data for all widgets.
   - Choose **Off** to temporarily turn off live data for all widgets.
   - Choose **Do not override** to preserve each widget's live data setting.

Using live data for a CloudWatch dashboard widget

To choose whether to use live data on a single widget

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. Select a widget, and choose **Actions**, **Edit**.
4. Choose the **Graph options** tab.
5. Select or clear the check box under **Live Data**.

Viewing an animated CloudWatch dashboard

You can view an animated dashboard that replays CloudWatch metric data that was captured over time. This can help you see trends, make presentations, or analyze issues after they occur.

Animated widgets in the dashboard include line widgets, stacked area widgets, number widgets, and metrics explorer widgets. Pie graphs, bar charts, text widgets, and logs widgets are displayed in the dashboard but are not animated.

To view an animated dashboard

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Dashboards**.
3. Choose the name of the dashboard.
4. Choose **Actions**, **Replay dashboard**.

5. (Optional) By default, when you start the animation, it appears as a sliding window. If you want the animation to appear as a point-by-point animation instead, choose the magnifying glass icon while the animation is paused and reset the zoom.
6. To start the animation, choose the Play button. You can also choose the back and forward buttons to move to other points in time.
7. (Optional) To change the time window for the animation, choose the calendar and select the time period.
8. To change the speed of the animation, choose **Auto speed** and select the new speed.
9. When you are finished, choose **Exit animate**.

Adding a CloudWatch dashboard to your favorites list

In the CloudWatch console, you can add dashboards, alarms, and log groups to a favorites list. You can access the favorites list from the navigation pane. The following procedure describes how to add a dashboard to the favorites list.

To add a dashboard to the favorites list

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Dashboards**.
3. From the list of dashboards, select the star symbol next to the name of the dashboard that you want to favorite.
   - (Optional) You also can favorite a dashboard by selecting a dashboard from the list and choosing the star symbol next to the dashboard name.
4. To access the favorites list, choose **Favorites and recents** in the navigation pane. The menu contains two columns. One contains your favorite dashboards, alarms, and log groups, and the other column contains the dashboards, alarms, and log groups that you recently visited.

> **Tip**
>
> You can favorite dashboards, as well as alarms and log groups, from the ***Favorites and recents*** menu in the CloudWatch console navigation pane. Under the ***Recently visited*** column, hover over the dashboard that you want to favorite, and choose the star symbol next to it.

Changing the period override setting or refresh interval for the CloudWatch dashboard

You can specify how the period setting of graphs added to this dashboard are retained or modified.

When an auto period or persisted time range is applied to a widget, the overall time range of the graph can affect the periods that you have set.

- If the time range is one day or less, period settings are not changed.
- If the time range is between one day and three days, periods set to below five minutes are changed to 5 minutes.
- If the time range is more than three days, periods set to below one hour are changed to one hour.

The following steps explain how to use the console to change the period override options. You can also change them by using the `periodOverride` field in the JSON structure of the dashboard. For more information, see [Dashboard Body Overall Structure](#).

To change the period override options

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. Choose **Actions**.
3. Under **Period override**, choose one of the following:

   - Choose **Auto** to have the period of the metrics on each graph automatically adapt to the dashboard's time range.
   - Choose **Do not override** to ensure that the period setting of each graph is always obeyed.
   - Choose one of the other options to cause graphs added to the dashboard to always adapt that chosen time as their period setting.

   The **Period override** always reverts to **Auto** when the dashboard is closed or the browser is refreshed. Different settings for **Period override** can't be saved.

You can change how often the data on your CloudWatch dashboard is refreshed.

To change the dashboard refresh interval

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. On the **Refresh options** menu (upper-right corner), choose **10 Seconds**, **1 Minute**, **2 Minutes**, **5 Minutes**, or **15 Minutes**.

Changing the time range or time zone format of a CloudWatch dashboard

You can change the time range to display dashboard data over minutes, hours, days, or weeks. You also can change the time zone format to display dashboard data in UTC or local time. Local time is the time zone that's specified in your computer's operating system.

> **Note**
>
> If you create a dashboard with graphs that contain 100 or more high-resolution metrics, we recommend that you don't set the time range to longer than 1 hour. For more information, see [High-resolution metrics](#).

> **Note**
>
> If the time range of a dashboard is shorter than the period used for a widget on the dashboard, the following happens:
>
> - The widget is modified to display the amount of data corresponding one complete period for that widget, even though this is longer than the dashboard time range. This ensures that there is at least one data point on the graph.
> - The start time of the period for this data point is adjusted backwards to ensure that at least one data point can be displayed.

New console

To change the dashboard time range

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. From the dashboard screen, do one of the following:
   - In the upper area of the dashboard, select one of the predefined time ranges. These span from 1 hour to 1 week (**1h**, **3h**, **12h**, **1d**, or **1w**).
   - Alternatively, you can choose one of the following custom time range options:
     - Choose **Custom**, and then choose the **Relative** tab. Choose a time range from 1 minute to 15 months.
     - Choose **Custom**, and then choose the **Absolute** tab. Use the calendar or text fields to specify your time range.

> **Tip**
>
> If the aggregation period is set to **Auto** when you change the time range of a graph, CloudWatch might change the period. To set the period manually, choose the **Actions** dropdown, and then choose **Period override**.

To change the dashboard time zone format

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. In the upper area of the dashboard, choose **Custom**.
4. In the upper-right corner of the box that appears, select **UTC** or **Local time** from the dropdown.
5. Choose **Apply**.

Old console

To change the dashboard time range

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. From the dashboard screen, do one of the following:
   - In the upper area of the dashboard, select one of the predefined time ranges. These span from 1 hour to 1 week (**1h**, **3h**, **12h**, **1d**, **3d**, or **1w**).
   - Alternatively, you can choose one of the following custom time range options:
     - Choose the **custom** dropdown, and then choose the **Relative** tab. Select one of the predefined ranges, which span from 1 minute to 15 months.
     - Choose the **custom** dropdown, and then choose the **Absolute** tab. Use the calendar or text fields to specify your time range.

> **Tip**
>
> If the aggregation period is set to **Auto** when you change the time range of a graph, CloudWatch might change the period. To set the period manually, choose the **Actions** dropdown, and then choose **Period override**.

To change the dashboard time zone format

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. In the upper-right corner of the dashboard screen, choose the **Custom** dropdown.
4. In the upper-right corner of the box that appears, select **UTC** or **Local timezone** from the dropdown.

Metrics in Amazon CloudWatch

Metrics are data about the performance of your systems. By default, many services provide free metrics for resources (such as Amazon EC2 instances, Amazon EBS volumes, and Amazon RDS DB instances). You can also enable detailed monitoring for some resources, such as your Amazon EC2 instances, or publish your own application metrics. Amazon CloudWatch can load all the metrics in your account (both Amazon resource metrics and application metrics that you provide) for search, graphing, and alarms.

Metric data is kept for 15 months, enabling you to view both up-to-the-minute data and historical data.

To graph metrics in the console, you can use CloudWatch Metrics Insights, a high-performance SQL query engine that you can use to identify trends and patterns within all your metrics in real time.

Metrics concepts

The following terminology and concepts are central to your understanding and use of Amazon CloudWatch:

- Namespaces
- Metrics
- Dimensions
- Resolution
- Statistics
- Percentiles
- Alarms

For information about the service quotas for CloudWatch metrics, alarms, API requests, and alarm email notifications, see CloudWatch service quotas.

Namespaces

A *namespace* is a container for CloudWatch metrics. Metrics in different namespaces are isolated from each other, so that metrics from different applications are not mistakenly aggregated into the same statistics.

There is no default namespace. You must specify a namespace for each data point you publish to CloudWatch. You can specify a namespace name when you create a metric. These names must contain valid ASCII characters, and be 255 or fewer characters. Possible characters are: alphanumeric characters (0-9A-Za-z), period (.), hyphen (-), underscore (_), forward slash (/), hash (#), colon (:), and the space character. A namespace must contain at least one non-whitespace character.

The Amazon namespaces typically use the following naming convention: `AWS/service`. For example, Amazon EC2 uses the `AWS/EC2` namespace. For the list of Amazon namespaces, see Amazon services that publish CloudWatch metrics.

Metrics

Metrics are the fundamental concept in CloudWatch. A metric represents a time-ordered set of data points that are published to CloudWatch. Think of a metric as a variable to monitor, and the data points as representing the values of that variable over time. For example, the CPU usage of a particular EC2 instance is one metric provided by Amazon EC2. The data points themselves can come from any application or business activity from which you collect data.

By default, many Amazon services provide metrics at no charge for resources (such as Amazon EC2 instances, Amazon EBS volumes, and Amazon RDS DB instances). For a charge, you can also enable detailed monitoring for some resources, such as your Amazon EC2 instances, or publish your own application metrics. For custom metrics, you can add the data points in any order, and at any rate you choose. You can retrieve statistics about those data points as an ordered set of time-series data.

Metrics exist only in the Region in which they are created. Metrics cannot be deleted, but they automatically expire after 15 months if no new data is published to them. Data points older than 15 months expire on a rolling basis; as new data points come in, data older than 15 months is dropped.

Metrics are uniquely defined by a name, a namespace, and zero or more dimensions. Each data point in a metric has a time stamp, and (optionally) a unit of measure. You can retrieve statistics from CloudWatch for any metric.

For more information, see View available metrics and Publish custom metrics.

Time stamps

Each metric data point must be associated with a time stamp. The time stamp can be up to two weeks in the past and up to two hours into the future. If you do not provide a time stamp, CloudWatch creates a time stamp for you based on the time the data point was received.

Time stamps are `dateTime` objects, with the complete date plus hours, minutes, and seconds (for example, 2016-10-31T23:59:59Z). For more information, see [dateTime](). Although it is not required, we recommend that you use Coordinated Universal Time (UTC). When you retrieve statistics from CloudWatch, all times are in UTC.

CloudWatch alarms check metrics based on the current time in UTC. Custom metrics sent to CloudWatch with time stamps other than the current UTC time can cause alarms to display the **Insufficient Data** state or result in delayed alarms.

Metrics retention

CloudWatch retains metric data as follows:

- Data points with a period of less than 60 seconds are available for 3 hours. These data points are high-resolution custom metrics.
- Data points with a period of 60 seconds (1 minute) are available for 15 days
- Data points with a period of 300 seconds (5 minutes) are available for 63 days
- Data points with a period of 3600 seconds (1 hour) are available for 455 days (15 months)

Data points that are initially published with a shorter period are aggregated together for long-term storage. For example, if you collect data using a period of 1 minute, the data remains available for 15 days with 1-minute resolution. After 15 days this data is still available, but is aggregated and is retrievable only with a resolution of 5 minutes. After 63 days, the data is further aggregated and is available with a resolution of 1 hour.

> **Note**
>
> Metrics that have not had any new data points in the past two weeks do not appear in the console. They also do not appear when you type their metric name or dimension names in the search box in the **All metrics** tab in the console, and they are not returned in the results of a list-metrics command. The best way to retrieve these metrics is with the get-metric-data or get-metric-statistics commands in the Amazon CLI.

Dimensions

A *dimension* is a name/value pair that is part of the identity of a metric. You can assign up to 30 dimensions to a metric.

Every metric has specific characteristics that describe it, and you can think of dimensions as categories for those characteristics. Dimensions help you design a structure for your statistics plan. Because dimensions are part of the unique identifier for a metric, whenever you add a unique name/value pair to one of your metrics, you are creating a new variation of that metric.

Amazon services that send data to CloudWatch attach dimensions to each metric. You can use dimensions to filter the results that CloudWatch returns. For example, you can get statistics for a specific EC2 instance by specifying the `InstanceId` dimension when you search for metrics.

For metrics produced by certain Amazon services, such as Amazon EC2, CloudWatch can aggregate data across dimensions. For example, if you search for metrics in the `AWS/EC2` namespace but do not specify any dimensions, CloudWatch aggregates all data for the specified metric to create the statistic that you requested. CloudWatch does not aggregate across dimensions for your custom metrics.

Dimension combinations

CloudWatch treats each unique combination of dimensions as a separate metric, even if the metrics have the same metric name. You can only retrieve statistics using combinations of dimensions that you specifically published. When you retrieve statistics, specify the same values for the namespace, metric name, and dimension parameters that were used when the metrics were created. You can also specify the start and end times for CloudWatch to use for aggregation.

For example, suppose that you publish four distinct metrics named ServerStats in the DataCenterMetric namespace with the following properties:

```
Dimensions: Server=Prod, Domain=Frankfurt, Unit: Count, Timestamp:
 2016-10-31T12:30:00Z, Value: 105
Dimensions: Server=Beta, Domain=Frankfurt, Unit: Count, Timestamp:
 2016-10-31T12:31:00Z, Value: 115
Dimensions: Server=Prod, Domain=Rio,       Unit: Count, Timestamp:
 2016-10-31T12:32:00Z, Value: 95
Dimensions: Server=Beta, Domain=Rio,       Unit: Count, Timestamp:
 2016-10-31T12:33:00Z, Value: 97
```

If you publish only those four metrics, you can retrieve statistics for these combinations of dimensions:

- `Server=Prod,Domain=Frankfurt`
- `Server=Prod,Domain=Rio`
- `Server=Beta,Domain=Frankfurt`
- `Server=Beta,Domain=Rio`

You can't retrieve statistics for the following dimensions or if you specify no dimensions. (The exception is by using the metric math **SEARCH** function, which can retrieve statistics for multiple metrics. For more information, see Use search expressions in graphs.)

- `Server=Prod`
- `Server=Beta`
- `Domain=Frankfurt`
- `Domain=Rio`

Resolution

Each metric is one of the following:

- Standard resolution, with data having a one-minute granularity
- High resolution, with data at a granularity of one second

Metrics produced by Amazon services are standard resolution by default. When you publish a custom metric, you can define it as either standard resolution or high resolution. When you publish a high-resolution metric, CloudWatch stores it with a resolution of 1 second, and you can read and retrieve it with a period of 1 second, 5 seconds, 10 seconds, 30 seconds, or any multiple of 60 seconds.

High-resolution metrics can give you more immediate insight into your application's sub-minute activity. Keep in mind that every `PutMetricData` call for a custom metric is charged, so calling `PutMetricData` more often on a high-resolution metric can lead to higher charges. For more information about CloudWatch pricing, see Amazon CloudWatch Pricing.

If you set an alarm on a high-resolution metric, you can specify a high-resolution alarm with a period of 10 seconds or 30 seconds, or you can set a regular alarm with a period of any multiple of 60 seconds. There is a higher charge for high-resolution alarms with a period of 10 or 30 seconds.

Statistics

Statistics are metric data aggregations over specified periods of time. CloudWatch provides statistics based on the metric data points provided by your custom data or provided by other Amazon services to CloudWatch. Aggregations are made using the namespace, metric name, dimensions, and the data point unit of measure, within the time period you specify.

For detailed definitions of the statistics supported by CloudWatch, see [CloudWatch statistics definitions](#).

Units

Each statistic has a unit of measure. Example units include `Bytes`, `Seconds`, `Count`, and `Percent`. For the complete list of the units that CloudWatch supports, see the [MetricDatum](#) data type in the *Amazon CloudWatch API Reference*.

You can specify a unit when you create a custom metric. If you do not specify a unit, CloudWatch uses None as the unit. Units help provide conceptual meaning to your data. Though CloudWatch attaches no significance to a unit internally, other applications can derive semantic information based on the unit.

Metric data points that specify a unit of measure are aggregated separately. When you get statistics without specifying a unit, CloudWatch aggregates all data points of the same unit together. If you have two otherwise identical metrics with different units, two separate data streams are returned, one for each unit.

Periods

A *period* is the length of time associated with a specific Amazon CloudWatch statistic. Each statistic represents an aggregation of the metrics data collected for a specified period of time. Periods are defined in numbers of seconds, and valid values for period are 1, 5, 10, 30, or any multiple of 60. For example, to specify a period of six minutes, use 360 as the period value. You can adjust how the data is aggregated by varying the length of the period. The default value of a period is 60 seconds. A period can be as short as one second, and must be a multiple of 60 if it is greater than the default value of 60 seconds.

Only custom metrics that you define with a storage resolution of 1 second support sub-minute periods. Even though the option to set a period below 60 is always available in the console, you should select a period that aligns to how the metric is stored. For more information about metrics that support sub-minute periods, see High-resolution metrics.

When you retrieve statistics, you can specify a period, start time, and end time. These parameters determine the overall length of time associated with the statistics. The default values for the start time and end time get you the last hour's worth of statistics. The values that you specify for the start time and end time determine how many periods CloudWatch returns. For example, retrieving statistics using the default values for the period, start time, and end time returns an aggregated set of statistics for each minute of the previous hour. If you prefer statistics aggregated in ten-minute blocks, specify a period of 600. For statistics aggregated over the entire hour, specify a period of 3600.

When statistics are aggregated over a period of time, they are stamped with the time corresponding to the beginning of the period. For example, data aggregated from 7:00pm to 8:00pm is stamped as 7:00pm. Additionally, data aggregated between 7:00pm and 8:00pm begins to be visible at 7:00pm, then the values of that aggregated data may change as CloudWatch collects more samples during the period.

Periods are also important for CloudWatch alarms. When you create an alarm to monitor a specific metric, you are asking CloudWatch to compare that metric to the threshold value that you specified. You have extensive control over how CloudWatch makes that comparison. Not only can you specify the period over which the comparison is made, but you can also specify how many evaluation periods are used to arrive at a conclusion. For example, if you specify three evaluation periods, CloudWatch compares a window of three data points. CloudWatch only notifies you if the oldest data point is breaching and the others are breaching or missing.

Aggregation

Amazon CloudWatch aggregates statistics according to the period length that you specify when retrieving statistics. You can publish as many data points as you want with the same or similar time stamps. CloudWatch aggregates them according to the specified period length. CloudWatch does not automatically aggregate data across Regions, but you can use metric math to aggregate metrics from different Regions.

You can publish data points for a metric that share not only the same time stamp, but also the same namespace and dimensions. CloudWatch returns aggregated statistics for those data points. You can also publish multiple data points for the same or different metrics, with any time stamp.

For large datasets, you can insert a pre-aggregated dataset called a *statistic set*. With statistic sets, you give CloudWatch the Min, Max, Sum, and SampleCount for a number of data points. This is commonly used when you need to collect data many times in a minute. For example, suppose you have a metric for the request latency of a web page. It doesn't make sense to publish data with every web page hit. We suggest that you collect the latency of all hits to that web page, aggregate them once a minute, and send that statistic set to CloudWatch.

Amazon CloudWatch doesn't differentiate the source of a metric. If you publish a metric with the same namespace and dimensions from different sources, CloudWatch treats this as a single metric. This can be useful for service metrics in a distributed, scaled system. For example, all the hosts in a web server application could publish identical metrics representing the latency of requests they are processing. CloudWatch treats these as a single metric, allowing you to get the statistics for minimum, maximum, average, and sum of all requests across your application.

Percentiles

A *percentile* indicates the relative standing of a value in a dataset. For example, the 95th percentile means that 95 percent of the data is lower than this value and 5 percent of the data is higher than this value. Percentiles help you get a better understanding of the distribution of your metric data.

Percentiles are often used to isolate anomalies. In a normal distribution, 95 percent of the data is within two standard deviations from the mean and 99.7 percent of the data is within three standard deviations from the mean. Any data that falls outside three standard deviations is often considered to be an anomaly because it differs so greatly from the average value. For example, suppose that you are monitoring the CPU utilization of your EC2 instances to ensure that your customers have a good experience. If you monitor the average, this can hide anomalies. If you monitor the maximum, a single anomaly can skew the results. Using percentiles, you can monitor the 95th percentile of CPU utilization to check for instances with an unusually heavy load.

Some CloudWatch metrics support percentiles as a statistic. For these metrics, you can monitor your system and applications using percentiles as you would when using the other CloudWatch statistics (Average, Minimum, Maximum, and Sum). For example, when you create an alarm, you can use percentiles as the statistical function. You can specify the percentile with up to ten decimal places (for example, p95.0123456789).

Percentile statistics are available for custom metrics as long as you publish the raw, unsummarized data points for your custom metric. Percentile statistics are not available for metrics when any of the metric values are negative numbers.

CloudWatch needs raw data points to calculate percentiles. If you publish data using a statistic set instead, you can only retrieve percentile statistics for this data if one of the following conditions is true:

- The SampleCount value of the statistic set is 1 and Min, Max, and Sum are all equal.
- The Min and Max are equal, and Sum is equal to Min multiplied by SampleCount.

The following Amazon services include metrics that support percentile statistics.

- API Gateway
- Application Load Balancer
- Amazon EC2
- Elastic Load Balancing
- Kinesis
- Lambda
- Amazon RDS

CloudWatch also supports trimmed mean and other performance statistics, which can have a similar use as percentiles. For more information, see CloudWatch statistics definitions.

Alarms

You can use an *alarm* to automatically initiate actions on your behalf. An alarm watches a single metric over a specified time period, and performs one or more specified actions, based on the value of the metric relative to a threshold over time. The action is a notification sent to an Amazon SNS topic or an Auto Scaling policy. You can also add alarms to dashboards.

Alarms invoke actions for sustained state changes only. CloudWatch alarms do not invoke actions simply because they are in a particular state. The state must have changed and been maintained for a specified number of periods.

When creating an alarm, select an alarm monitoring period that is greater than or equal to the metrics resolution. For example, basic monitoring for Amazon EC2 provides metrics for your instances every 5 minutes. When setting an alarm on a basic monitoring metric, select a period of at least 300 seconds (5 minutes). Detailed monitoring for Amazon EC2 provides metrics for your instances with a resolution of 1 minute. When setting an alarm on a detailed monitoring metric, select a period of at least 60 seconds (1 minute).

If you set an alarm on a high-resolution metric, you can specify a high-resolution alarm with a period of 10 seconds or 30 seconds, or you can set a regular alarm with a period of any multiple of 60 seconds. There is a higher charge for high-resolution alarms. For more information about high-resolution metrics, see Publish custom metrics.

For more information, see Using Amazon CloudWatch alarms and Create an alarm from a metric on a graph.

Basic monitoring and detailed monitoring in CloudWatch

CloudWatch provides two categories of monitoring: *basic monitoring* and *detailed monitoring*.

Many Amazon services offer basic monitoring by publishing a default set of metrics to CloudWatch with no charge to customers. By default, when you start using one of these Amazon Web Services services, basic monitoring is automatically enabled. For a list of services that offer basic monitoring, see Amazon services that publish CloudWatch metrics.

Detailed monitoring is offered by only some services. It also incurs charges. To use it for an Amazon service, you must choose to activate it. For more information about pricing, see Amazon CloudWatch pricing.

Detailed monitoring options differ based on the services that offer it. For example, Amazon EC2 detailed monitoring provides more frequent metrics, published at one-minute intervals, instead of the five-minute intervals used in Amazon EC2 basic monitoring. Detailed monitoring for Amazon S3 and Amazon Managed Streaming for Apache Kafka means more fine-grained metrics.

In different Amazon services, detailed monitoring also has different names. For example, in Amazon EC2 it is called detailed monitoring, in Amazon Elastic Beanstalk it is called enhanced monitoring, and in Amazon S3 it is called request metrics.

Using detailed monitoring for Amazon EC2 helps you better manage your Amazon EC2 resources, so that you can find trends and take action faster. For Amazon S3 request metrics are available at one-minute intervals to help you quickly identify and act on operational issues. On Amazon MSK, when you enable the `PER_BROKER`, `PER_TOPIC_PER_BROKER`, or `PER_TOPIC_PER_PARTITION` level monitoring, you get additional metrics that provide more visibility.

The following table lists the services that offer detailed monitoring. It also includes links to the documentation for those services that explain more about the detailed monitoring and provide instructions for how to activate it.

Service	Documentation
Amazon API Gateway	Dimensions for API Gateway metrics
Amazon AppSync	CloudWatch metrics
Amazon CloudFront	Viewing additional CloudFront distribution metrics
Amazon EC2	Manage detailed monitoring for your EC2 instances
Elastic Beanstalk	Enhanced health reporting and monitoring
Amazon Kinesis Data Streams	Enhanced Shard-level Metrics
Amazon Lambda	Event source mapping metrics
Amazon MSK	Amazon MSK Metrics for Monitoring with CloudWatch
Amazon S3	Amazon S3 request metrics in CloudWatch

Service	Documentation
Amazon SES	Collect CloudWatch detailed monitoring metrics using Amazon SES event publishing.

Additionally, CloudWatch offers out-of-the-box monitoring solutions with more detailed metrics and pre-created dashboards for some Amazon services, as shown in the following table.

Service	Feature documentation
Lambda	Lambda Insights
Amazon ECS	Container Insights for Amazon ECS
Amazon EKS	Container Insights for Amazon EKS and Kubernetes
Amazon RDS	CloudWatch Database Insights
Amazon Aurora	CloudWatch Database Insights

Query your CloudWatch metrics with CloudWatch Metrics Insights

CloudWatch Metrics Insights is a powerful high-performance SQL query engine that you can use to query your metrics at scale. You can identify trends and patterns within all of your CloudWatch metrics in real time.

You can also set alarms on any Metrics Insights queries that return a single time series. This can be especially useful to create alarms that watch aggregated metrics across a fleet of your infrastructure or applications. Create the alarm once, and it dynamically adjusts as resources are added to or removed from the fleet.

You can perform a CloudWatch Metrics Insights query in the console with the CloudWatch Metrics Insights query editor. You can also perform a CloudWatch Metrics Insights query with the Amazon CLI or an Amazon SDK by running `GetMetricData` or `PutDashboard`. There's no charge for queries that you run with the CloudWatch Metrics Insights query editor. For more information about CloudWatch pricing, see [Amazon CloudWatch Pricing](#).

With the CloudWatch Metrics Insights query editor, you can choose from a variety of prebuilt sample queries and also create your own queries. As you create your queries, you can use a builder view to browse your existing metrics and dimensions. Alternatively, use an editor view to manually write queries.

You can also use natural language to create CloudWatch Metrics Insights queries. To do so, ask questions about or describe the data you're looking for. This AI-assisted capability generates a query based on your prompt and provides a line-by-line explanation of how the query works. For more information, see [Use natural language to generate and update CloudWatch Metrics Insights queries](#).

With Metrics Insights, you can run queries at scale. With the **GROUP BY** clause, you can group your metrics in real time into separate time series per specific dimension value. Because Metrics Insights queries include an **ORDER BY** ability, you can use Metrics Insights to make "Top N" type queries. For example, "Top N" type queries can scan millions of metrics in your account and return the 10 instances that consume the most CPU. This can help you pinpoint and remedy latency issues in your applications.

Topics

- Query components and syntax in CloudWatch Metrics Insights
- Alarms on CloudWatch Metrics Insights queries in CloudWatch
- Use Metrics Insights queries with metric math
- Use natural language to generate and update CloudWatch Metrics Insights queries
- SQL inference
- Metrics Insights sample queries
- Metrics Insights limits
- Metrics Insights glossary
- Troubleshooting Metrics Insights

Building your queries in CloudWatch Metrics Insights

You can run a CloudWatch Metrics Insights query using the CloudWatch console, the Amazon CLI, or the Amazon SDKs. Queries run in the console are free of charge. For more information about CloudWatch pricing, see Amazon CloudWatch Pricing.

For more information about using the Amazon SDKs to perform a Metrics Insights query, see GetMetricData.

To run a query using the CloudWatch console, follow these steps:

To query your metrics using Metrics Insights

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Metrics**, **All metrics**.
3. (Optional) To run a pre-built sample query, choose **Add query** and select the query to run. If you are satisfied with this query, you can skip the rest of this procedure. Or, you can choose **Editor** to edit the sample query and then choose **Run** to run the modified query.
4. To create your own query, choose **Multi source query**. You can then use the **Builder** view, the **Editor** view, and also use a combination of both. You can switch between the two views anytime and see your work in progress in both views.

   In the **Builder** view, you can browse and select the metric namespace, metric name, filter, group, and order options. For each of these options, the query builder offers you a list of possible selections from your environment to choose from.

In the **Editor** view, you can start writing your query. As you type, the editor offers suggestions based on the characters that you have typed so far.

5. When you are satisfied with your query, choose **Run**.
6. (Optional) Another way to edit a query that you have graphed is to choose the **Graphed metrics** tab and choose the edit icon next to the query formula in the **Details** column.
7. (Optional) To remove a query from the graph, choose **Graphed metrics** and choose the **X** icon at the right side of the row that displays your query.

Query components and syntax in CloudWatch Metrics Insights

CloudWatch Metrics Insights syntax is as follows.

```
SELECT FUNCTION(metricName)
FROM namespace | SCHEMA(...)
[ WHERE labelKey OPERATOR labelValue [AND ... ] ]
[ GROUP BY labelKey [ , ... ] ]
[ ORDER BY FUNCTION() [ DESC | ASC ] ]
[ LIMIT number ]
```

The possible clauses in a Metrics Insights query are as follows. None of the keywords are case sensitive, but the identifiers such as the names of metrics, namespaces, and dimensions are case sensitive.

SELECT

Required. Specifies the function to use to aggregate observations in each time bucket (determined by the provided period). Also specifies the name of the metric to query.

The valid values for **FUNCTION** are `AVG`, `COUNT`, `MAX`, `MIN`, and `SUM`.

- `AVG` calculates the average of the observations matched by the query.
- `COUNT` returns the count of the observations matched by the query.
- `MAX` returns the maximum value of the observations matched by the query.
- `MIN` returns the minimum value of the observations matched by the query.
- `SUM` calculates the sum of the observations matched by the query.

FROM

Required. Specifies the source of the metric. You can specify either the metric namespace that contains the metric that is to be queried, or a **SCHEMA** table function. Examples of metric namespaces include `"AWS/EC2"`, `"AWS/Lambda"`, and metric namespaces that you have created for your custom metrics.

Metric namespaces that include **/** or any other character that is not a letter, number, or underscore must be surrounded by double quotation marks. For more information, see [What needs quotation marks or escape characters?](#).

SCHEMA

An optional table function that can be used within a **FROM** clause. Use **SCHEMA** to scope down the query results to only the metrics that exactly match a list of dimensions, or to metrics that have no dimensions.

If you use a **SCHEMA** clause, it must contain at least one argument, and this first argument must be the metric namespace being queried. If you specify **SCHEMA** with only this namespace argument, the results are scoped down to only metrics that do not have any dimensions.

If you specify **SCHEMA** with additional arguments, the additional arguments after the namespace argument must be *label* keys. Label keys must be dimension names. If you specify one or more of these label keys, the results are scoped down to only those metrics that have that exact set of dimensions. The order of these label keys does not matter.

For example:

- **SELECT AVG(CPUUtilization) FROM "AWS/EC2"** matches all `CPUUtilization` metrics in the `AWS/EC2` namespace, no matter their dimensions, and returns a single aggregated time series.
- **SELECT AVG(CPUUtilization) FROM SCHEMA("AWS/EC2")** matches only the `CPUUtilization` metrics in the `AWS/EC2` namespace that do not have any dimensions defined.
- **SELECT AVG(CPUUtilization) FROM SCHEMA("AWS/EC2", InstanceId)** matches only the `CPUUtilization` metrics that were reported to CloudWatch with exactly one dimension, `InstanceId`.
- **SELECT SUM(RequestCount) FROM SCHEMA("AWS/ApplicationELB", LoadBalancer, AvailabilityZone)** matches only the `RequestCount` metrics that were reported to

CloudWatch from `AWS/ApplicationELB` with exactly two dimensions, `LoadBalancer` and `AvailabilityZone`.

WHERE

Optional. Filters the results to only those metrics that match your specified expression using specific label values for one or more label keys. For example, **WHERE InstanceType = 'c3.4xlarge'** filters the results to only `c3.4xlarge` instance types, and **WHERE InstanceType != 'c3.4xlarge'** filters the results to all instance types except `c3.4xlarge`.

When you run a query in a monitoring account, you can use `WHERE AWS.AccountId` to limit results to only the account that you specify. For example, `WHERE AWS.AccountId=444455556666` queries metrics from only account 444455556666. To limit your query to only metrics in the monitoring account itself, use `WHERE AWS.AccountId=CURRENT_ACCOUNT_ID()`.

Label values must always be enclosed with single quotation marks.

Supported operators

The **WHERE** clause supports the following operators:

- **=** Label value must match the specified string.
- **!=** Label value must not match the specified string.
- **AND** Both conditions that are specified must be true to match. You can use multiple **AND** keywords to specify two or more conditions.

GROUP BY

Optional. Groups the query results into multiple time series, each one corresponding to a different value for the specified label key or keys. For example, using `GROUP BY InstanceId` returns a different time series for each value of `InstanceId`. Using `GROUP BY ServiceName, Operation` creates a different time series for each possible combination of the values of `ServiceName` and `Operation`.

With a **GROUP BY** clause, by default the results are ordered in alphabetical ascending order, using the sequence of labels specified in the **GROUP BY** clause. To change the order of the results, add an **ORDER BY** clause to your query.

When you run a query in a monitoring account, you can use `GROUP BY AWS.AccountId` to group the results based on the accounts they are from.

> **Note**
>
> If some of the matching metrics don't include a specific label key specified in the **GROUP BY** clause, a null group named `Other` is returned. For example, if you specify `GROUP BY ServiceName, Operation` and some of the returned metrics don't include `ServiceName` as a dimension, then those metrics are displayed as having `Other` as the value for `ServiceName`.

ORDER BY

Optional. Specifies the order to use for the returned time series, if the query returns more than one time series. The order is based on the values found by the **FUNCTION** that you specify in the **ORDER BY** clause. The **FUNCTION** is used to calculate a single scalar value from each returned time series, and that value is used to determine the order.

You also specify whether to use ascending **ASC** or descending **DESC** order. If you omit this, the default is ascending **ASC**.

For example, adding an `ORDER BY MAX() DESC` clause orders the results by the maximum data point observed within the time range, in descending order: meaning that the time series that has the highest maximum data point is returned first.

The valid functions to use within an **ORDER BY** clause are `AVG()`, `COUNT()`, `MAX()`, `MIN()`, and `SUM()`.

If you use an **ORDER BY** clause with a **LIMIT** clause, the resulting query is a "Top N" query. **ORDER BY** is also useful for queries that might return a large number of metrics, because each query can return no more than 500 time series. If a query matches more than 500 time series, and you use an **ORDER BY** clause, the time series are sorted and then the 500 time series that come first in the sort order are the ones that are returned.

LIMIT

Optional. Limits the number of time series returned by the query to the value that you specify. The maximum value that you can specify is 500, and a query that does not specify a **LIMIT** can also return no more than 500 time series.

Using a **LIMIT** clause with an **ORDER BY** clause gives you a "Top N" query.

What needs quotation marks or escape characters?

In a query, label values must always be surrounded with single quotation marks. For example, **SELECT MAX(CPUUtilization) FROM "AWS/EC2" WHERE AutoScalingGroupName = 'my-production-fleet'**.

Metric namespaces, metric names, and label keys that contain characters other than letters, numbers, and underscore (_) must be surrounded by double quote marks. For example, **SELECT MAX("My.Metric")**.

If one of these contains a double quotation mark or single quotation mark itself (such as `Bytes"Input"`), you must escape each quotation mark with a backslash, as in **SELECT AVG("Bytes \"Input\"")**.

If a metric namespace, metric name, or label key, contains a word that is a reserved keyword in Metrics Insights, these must also be enclosed in double quotation marks. For example, if you have a metric named `LIMIT`, you would use `SELECT AVG("LIMIT")`. It is also valid to enclose any namespace, metric name, or label in double quotation marks even if it does not include a reserved keyword.

For a complete list of reserved keywords, see Reserved keywords.

Build a rich query step by step

This section illustrates building a full example that uses all possible clauses, step by step.

We start with the following query, which aggregates all of the Application Load Balancer `RequestCount` metrics that are collected with both the dimensions `LoadBalancer` and `AvailabilityZone`.

```
SELECT SUM(RequestCount)
FROM SCHEMA("AWS/ApplicationELB", LoadBalancer, AvailabilityZone)
```

Now, if we want to see metrics only from a specific load balancer, we can add a **WHERE** clause to limit the metrics returned to only those metrics where the value of the `LoadBalancer` dimension is `app/load-balancer-1`.

```
SELECT SUM(RequestCount)
FROM SCHEMA("AWS/ApplicationELB", LoadBalancer, AvailabilityZone)
WHERE LoadBalancer = 'app/load-balancer-1'
```

The preceding query aggregates the `RequestCount` metrics from all Availability Zones for this load balancer into one time series. If we want to see different time series for each Availability Zone, we can add a **GROUP BY** clause.

```
SELECT SUM(RequestCount)
FROM SCHEMA("AWS/ApplicationELB", LoadBalancer, AvailabilityZone)
WHERE LoadBalancer = 'app/load-balancer-1'
GROUP BY AvailabilityZone
```

Next, we might want to order these results to see the highest values first. The following **ORDER BY** clause orders the time series in descending order, by the maximum value reported by each time series during the query time range:

```
SELECT SUM(RequestCount)
FROM SCHEMA("AWS/ApplicationELB", LoadBalancer, AvailabilityZone)
WHERE LoadBalancer = 'app/load-balancer-1'
GROUP BY AvailabilityZone
ORDER BY MAX() DESC
```

Finally, if we are primarily interested in a "Top N" type of query, we can use a **LIMIT** clause. This final example limits the results to only the time series with the five highest `MAX` values.

```
SELECT SUM(RequestCount)
FROM SCHEMA("AWS/ApplicationELB", LoadBalancer, AvailabilityZone)
WHERE LoadBalancer = 'app/load-balancer-1'
GROUP BY AvailabilityZone
ORDER BY MAX() DESC
LIMIT 5
```

Cross-account query examples

These examples are valid when run in an account set up as a monitoring account in CloudWatch cross-account observability.

The following example searches all Amazon EC2 instances in the source account 123456789012 and returns the average.

```
SELECT AVG(CpuUtilization)
FROM "AWS/EC2"
WHERE AWS.AccountId ='123456789012'
```

The following example queries the `CPUUtilization` metric in `AWS/EC2` in all the linked source accounts, and groups the results by account ID and instance type.

```
SELECT AVG(CpuUtilization)
FROM "AWS/EC2"
GROUP BY AWS.AccountId, InstanceType
```

The following example queries the `CPUUtilization` in the monitoring account itself.

```
SELECT AVG(CpuUtilization)
FROM "AWS/EC2"
WHERE AWS.AccountId = CURRENT_ACCOUNT_ID()
```

Reserved keywords

The following are reserved keywords in CloudWatch Metrics Insights. If any of these words are in a namespace, metric name, or label key in a query, you must enclose them in double quote marks. Reserved keywords are not case sensitive.

```
"ABORT" "ABORTSESSION" "ABS" "ABSOLUTE" "ACCESS" "ACCESSIBLE" "ACCESS_LOCK" "ACCOUNT"
 "ACOS" "ACOSH" "ACTION" "ADD" "ADD_MONTHS"
"ADMIN" "AFTER" "AGGREGATE" "ALIAS" "ALL" "ALLOCATE" "ALLOW" "ALTER" "ALTERAND" "AMP"
 "ANALYSE" "ANALYZE" "AND" "ANSIDATE" "ANY" "ARE" "ARRAY",
"ARRAY_AGG" "ARRAY_EXISTS" "ARRAY_MAX_CARDINALITY" "AS" "ASC" "ASENSITIVE" "ASIN"
 "ASINH" "ASSERTION" "ASSOCIATE" "ASUTIME" "ASYMMETRIC" "AT",
"ATAN" "ATAN2" "ATANH" "ATOMIC" "AUDIT" "AUTHORIZATION" "AUX" "AUXILIARY" "AVE"
 "AVERAGE" "AVG" "BACKUP" "BEFORE" "BEGIN" "BEGIN_FRAME" "BEGIN_PARTITION",
"BETWEEN" "BIGINT" "BINARY" "BIT" "BLOB" "BOOLEAN" "BOTH" "BREADTH" "BREAK" "BROWSE"
 "BT" "BUFFERPOOL" "BULK" "BUT" "BY" "BYTE" "BYTEINT" "BYTES" "CALL",
"CALLED" "CAPTURE" "CARDINALITY" "CASCADE" "CASCADED" "CASE" "CASESPECIFIC" "CASE_N"
 "CAST" "CATALOG" "CCSID" "CD" "CEIL" "CEILING" "CHANGE" "CHAR",
"CHAR2HEXINT" "CHARACTER" "CHARACTERS" "CHARACTER_LENGTH" "CHARS" "CHAR_LENGTH" "CHECK"
 "CHECKPOINT" "CLASS" "CLASSIFIER" "CLOB" "CLONE" "CLOSE" "CLUSTER",
"CLUSTERED" "CM" "COALESCE" "COLLATE" "COLLATION" "COLLECT" "COLLECTION" "COLLID"
 "COLUMN" "COLUMN_VALUE" "COMMENT" "COMMIT" "COMPLETION" "COMPRESS" "COMPUTE",
"CONCAT" "CONCURRENTLY" "CONDITION" "CONNECT" "CONNECTION" "CONSTRAINT" "CONSTRAINTS"
 "CONSTRUCTOR" "CONTAINS" "CONTAINSTABLE" "CONTENT" "CONTINUE" "CONVERT",
"CONVERT_TABLE_HEADER" "COPY" "CORR" "CORRESPONDING" "COS" "COSH" "COUNT" "COVAR_POP"
 "COVAR_SAMP" "CREATE" "CROSS" "CS" "CSUM" "CT" "CUBE" "CUME_DIST",
"CURRENT" "CURRENT_CATALOG" "CURRENT_DATE" "CURRENT_DEFAULT_TRANSFORM_GROUP"
 "CURRENT_LC_CTYPE" "CURRENT_PATH" "CURRENT_ROLE" "CURRENT_ROW" "CURRENT_SCHEMA",
```

```
"CURRENT_SERVER" "CURRENT_TIME" "CURRENT_TIMESTAMP" "CURRENT_TIMEZONE"
 "CURRENT_TRANSFORM_GROUP_FOR_TYPE" "CURRENT_USER" "CURRVAL" "CURSOR" "CV" "CYCLE"
 "DATA",
"DATABASE" "DATABASES" "DATABLOCKSIZE" "DATE" "DATEFORM" "DAY" "DAYS" "DAY_HOUR"
 "DAY_MICROSECOND" "DAY_MINUTE" "DAY_SECOND" "DBCC" "DBINFO" "DEALLOCATE" "DEC",
"DECFLOAT" "DECIMAL" "DECLARE" "DEFAULT" "DEFERRABLE" "DEFERRED" "DEFINE" "DEGREES"
 "DEL" "DELAYED" "DELETE" "DENSE_RANK" "DENY" "DEPTH" "DEREF" "DESC" "DESCRIBE",
"DESCRIPTOR" "DESTROY" "DESTRUCTOR" "DETERMINISTIC" "DIAGNOSTIC" "DIAGNOSTICS"
 "DICTIONARY" "DISABLE" "DISABLED" "DISALLOW" "DISCONNECT" "DISK" "DISTINCT",
"DISTINCTROW" "DISTRIBUTED" "DIV" "DO" "DOCUMENT" "DOMAIN" "DOUBLE" "DROP" "DSSIZE"
 "DUAL" "DUMP" "DYNAMIC" "EACH" "ECHO" "EDITPROC" "ELEMENT" "ELSE" "ELSEIF",
"EMPTY" "ENABLED" "ENCLOSED" "ENCODING" "ENCRYPTION" "END" "END-EXEC" "ENDING"
 "END_FRAME" "END_PARTITION" "EQ" "EQUALS" "ERASE" "ERRLVL" "ERROR" "ERRORFILES",
"ERRORTABLES" "ESCAPE" "ESCAPED" "ET" "EVERY" "EXCEPT" "EXCEPTION" "EXCLUSIVE" "EXEC"
 "EXECUTE" "EXISTS" "EXIT" "EXP" "EXPLAIN" "EXTERNAL" "EXTRACT" "FALLBACK
"FALSE" "FASTEXPORT" "FENCED" "FETCH" "FIELDPROC" "FILE" "FILLFACTOR" "FILTER" "FINAL"
 "FIRST" "FIRST_VALUE" "FLOAT" "FLOAT4" "FLOAT8" "FLOOR"
"FOR" "FORCE" "FOREIGN" "FORMAT" "FOUND" "FRAME_ROW" "FREE" "FREESPACE" "FREETEXT"
 "FREETEXTTABLE" "FREEZE" "FROM" "FULL" "FULLTEXT" "FUNCTION"
"FUSION" "GE" "GENERAL" "GENERATED" "GET" "GIVE" "GLOBAL" "GO" "GOTO" "GRANT" "GRAPHIC"
 "GROUP" "GROUPING" "GROUPS" "GT" "HANDLER" "HASH"
"HASHAMP" "HASHBAKAMP" "HASHBUCKET" "HASHROW" "HAVING" "HELP" "HIGH_PRIORITY" "HOLD"
 "HOLDLOCK" "HOUR" "HOURS" "HOUR_MICROSECOND" "HOUR_MINUTE"
"HOUR_SECOND" "IDENTIFIED" "IDENTITY" "IDENTITYCOL" "IDENTITY_INSERT" "IF" "IGNORE"
 "ILIKE" "IMMEDIATE" "IN" "INCLUSIVE" "INCONSISTENT" "INCREMENT"
"INDEX" "INDICATOR" "INFILE" "INHERIT" "INITIAL" "INITIALIZE" "INITIALLY" "INITIATE"
 "INNER" "INOUT" "INPUT" "INS" "INSENSITIVE" "INSERT" "INSTEAD"
"INT" "INT1" "INT2" "INT3" "INT4" "INT8" "INTEGER" "INTEGERDATE" "INTERSECT"
 "INTERSECTION" "INTERVAL" "INTO" "IO_AFTER_GTIDS" "IO_BEFORE_GTIDS"
"IS" "ISNULL" "ISOBID" "ISOLATION" "ITERATE" "JAR" "JOIN" "JOURNAL" "JSON_ARRAY"
 "JSON_ARRAYAGG" "JSON_EXISTS" "JSON_OBJECT" "JSON_OBJECTAGG"
"JSON_QUERY" "JSON_TABLE" "JSON_TABLE_PRIMITIVE" "JSON_VALUE" "KEEP" "KEY" "KEYS"
 "KILL" "KURTOSIS" "LABEL" "LAG" "LANGUAGE" "LARGE" "LAST"
"LAST_VALUE" "LATERAL" "LC_CTYPE" "LE" "LEAD" "LEADING" "LEAVE" "LEFT" "LESS" "LEVEL"
 "LIKE" "LIKE_REGEX" "LIMIT" "LINEAR" "LINENO" "LINES"
"LISTAGG" "LN" "LOAD" "LOADING" "LOCAL" "LOCALE" "LOCALTIME" "LOCALTIMESTAMP" "LOCATOR"
 "LOCATORS" "LOCK" "LOCKING" "LOCKMAX" "LOCKSIZE" "LOG"
"LOG10" "LOGGING" "LOGON" "LONG" "LONGBLOB" "LONGTEXT" "LOOP" "LOWER" "LOW_PRIORITY"
 "LT" "MACRO" "MAINTAINED" "MAP" "MASTER_BIND"
"MASTER_SSL_VERIFY_SERVER_CERT" "MATCH" "MATCHES" "MATCH_NUMBER" "MATCH_RECOGNIZE"
 "MATERIALIZED" "MAVG" "MAX" "MAXEXTENTS" "MAXIMUM" "MAXVALUE"
"MCHARACTERS" "MDIFF" "MEDIUMBLOB" "MEDIUMINT" "MEDIUMTEXT" "MEMBER" "MERGE" "METHOD"
 "MICROSECOND" "MICROSECONDS" "MIDDLEINT" "MIN" "MINDEX"
```

```
"MINIMUM" "MINUS" "MINUTE" "MINUTES" "MINUTE_MICROSECOND" "MINUTE_SECOND" "MLINREG"
 "MLOAD" "MLSLABEL" "MOD" "MODE" "MODIFIES" "MODIFY"
"MODULE" "MONITOR" "MONRESOURCE" "MONSESSION" "MONTH" "MONTHS" "MSUBSTR" "MSUM"
 "MULTISET" "NAMED" "NAMES" "NATIONAL" "NATURAL" "NCHAR" "NCLOB"
"NE" "NESTED_TABLE_ID" "NEW" "NEW_TABLE" "NEXT" "NEXTVAL" "NO" "NOAUDIT" "NOCHECK"
 "NOCOMPRESS" "NONCLUSTERED" "NONE" "NORMALIZE" "NOT" "NOTNULL"
"NOWAIT" "NO_WRITE_TO_BINLOG" "NTH_VALUE" "NTILE" "NULL" "NULLIF" "NULLIFZERO" "NULLS"
 "NUMBER" "NUMERIC" "NUMPARTS" "OBID" "OBJECT" "OBJECTS"
"OCCURRENCES_REGEX" "OCTET_LENGTH" "OF" "OFF" "OFFLINE" "OFFSET" "OFFSETS" "OLD"
 "OLD_TABLE" "OMIT" "ON" "ONE" "ONLINE" "ONLY" "OPEN" "OPENDATASOURCE"
"OPENQUERY" "OPENROWSET" "OPENXML" "OPERATION" "OPTIMIZATION" "OPTIMIZE"
 "OPTIMIZER_COSTS" "OPTION" "OPTIONALLY" "OR" "ORDER" "ORDINALITY" "ORGANIZATION"
"OUT" "OUTER" "OUTFILE" "OUTPUT" "OVER" "OVERLAPS" "OVERLAY" "OVERRIDE" "PACKAGE" "PAD"
 "PADDED" "PARAMETER" "PARAMETERS" "PART" "PARTIAL" "PARTITION"
"PARTITIONED" "PARTITIONING" "PASSWORD" "PATH" "PATTERN" "PCTFREE" "PER" "PERCENT"
 "PERCENTILE" "PERCENTILE_CONT" "PERCENTILE_DISC" "PERCENT_RANK" "PERIOD" "PERM"
"PERMANENT" "PIECESIZE" "PIVOT" "PLACING" "PLAN" "PORTION" "POSITION" "POSITION_REGEX"
 "POSTFIX" "POWER" "PRECEDES" "PRECISION" "PREFIX" "PREORDER"
"PREPARE" "PRESERVE" "PREVVAL" "PRIMARY" "PRINT" "PRIOR" "PRIQTY" "PRIVATE"
 "PRIVILEGES" "PROC" "PROCEDURE" "PROFILE" "PROGRAM" "PROPORTIONAL"
"PROTECTION" "PSID" "PTF" "PUBLIC" "PURGE" "QUALIFIED" "QUALIFY" "QUANTILE" "QUERY"
 "QUERYNO" "RADIANS" "RAISERROR" "RANDOM" "RANGE" "RANGE_N" "RANK"
"RAW" "READ" "READS" "READTEXT" "READ_WRITE" "REAL" "RECONFIGURE" "RECURSIVE" "REF"
 "REFERENCES" "REFERENCING" "REFRESH" "REGEXP" "REGR_AVGX" "REGR_AVGY"
"REGR_COUNT" "REGR_INTERCEPT" "REGR_R2" "REGR_SLOPE" "REGR_SXX" "REGR_SXY" "REGR_SYY"
 "RELATIVE" "RELEASE" "RENAME" "REPEAT" "REPLACE" "REPLICATION"
"REPOVERRIDE" "REQUEST" "REQUIRE" "RESIGNAL" "RESOURCE" "RESTART" "RESTORE" "RESTRICT"
 "RESULT" "RESULT_SET_LOCATOR" "RESUME" "RET" "RETRIEVE" "RETURN"
"RETURNING" "RETURNS" "REVALIDATE" "REVERT" "REVOKE" "RIGHT" "RIGHTS" "RLIKE" "ROLE"
 "ROLLBACK" "ROLLFORWARD" "ROLLUP" "ROUND_CEILING" "ROUND_DOWN"
"ROUND_FLOOR" "ROUND_HALF_DOWN" "ROUND_HALF_EVEN" "ROUND_HALF_UP" "ROUND_UP" "ROUTINE"
 "ROW" "ROWCOUNT" "ROWGUIDCOL" "ROWID" "ROWNUM" "ROWS" "ROWSET"
"ROW_NUMBER" "RULE" "RUN" "RUNNING" "SAMPLE" "SAMPLEID" "SAVE" "SAVEPOINT" "SCHEMA"
 "SCHEMAS" "SCOPE" "SCRATCHPAD" "SCROLL" "SEARCH" "SECOND" "SECONDS"
"SECOND_MICROSECOND" "SECQTY" "SECTION" "SECURITY" "SECURITYAUDIT" "SEEK" "SEL"
 "SELECT" "SEMANTICKEYPHRASETABLE" "SEMANTICSIMILARITYDETAILSTABLE"
"SEMANTICSIMILARITYTABLE" "SENSITIVE" "SEPARATOR" "SEQUENCE" "SESSION" "SESSION_USER"
 "SET" "SETRESRATE" "SETS" "SETSESSRATE" "SETUSER" "SHARE" "SHOW"
"SHUTDOWN" "SIGNAL" "SIMILAR" "SIMPLE" "SIN" "SINH" "SIZE" "SKEW" "SKIP" "SMALLINT"
 "SOME" "SOUNDEX" "SOURCE" "SPACE" "SPATIAL" "SPECIFIC" "SPECIFICTYPE"
"SPOOL" "SQL" "SQLEXCEPTION" "SQLSTATE" "SQLTEXT" "SQLWARNING" "SQL_BIG_RESULT"
 "SQL_CALC_FOUND_ROWS" "SQL_SMALL_RESULT" "SQRT" "SS" "SSL" "STANDARD"
"START" "STARTING" "STARTUP" "STAT" "STATE" "STATEMENT" "STATIC" "STATISTICS" "STAY"
 "STDDEV_POP" "STDDEV_SAMP" "STEPINFO" "STOGROUP" "STORED" "STORES"
```

```
"STRAIGHT_JOIN" "STRING_CS" "STRUCTURE" "STYLE" "SUBMULTISET" "SUBSCRIBER" "SUBSET"
 "SUBSTR" "SUBSTRING" "SUBSTRING_REGEX" "SUCCEEDS" "SUCCESSFUL"
"SUM" "SUMMARY" "SUSPEND" "SYMMETRIC" "SYNONYM" "SYSDATE" "SYSTEM" "SYSTEM_TIME"
 "SYSTEM_USER" "SYSTIMESTAMP" "TABLE" "TABLESAMPLE" "TABLESPACE" "TAN"
"TANH" "TBL_CS" "TEMPORARY" "TERMINATE" "TERMINATED" "TEXTSIZE" "THAN" "THEN"
 "THRESHOLD" "TIME" "TIMESTAMP" "TIMEZONE_HOUR" "TIMEZONE_MINUTE" "TINYBLOB"
"TINYINT" "TINYTEXT" "TITLE" "TO" "TOP" "TRACE" "TRAILING" "TRAN" "TRANSACTION"
 "TRANSLATE" "TRANSLATE_CHK" "TRANSLATE_REGEX" "TRANSLATION" "TREAT"
"TRIGGER" "TRIM" "TRIM_ARRAY" "TRUE" "TRUNCATE" "TRY_CONVERT" "TSEQUAL" "TYPE" "UC"
 "UESCAPE" "UID" "UNDEFINED" "UNDER" "UNDO" "UNION" "UNIQUE"
"UNKNOWN" "UNLOCK" "UNNEST" "UNPIVOT" "UNSIGNED" "UNTIL" "UPD" "UPDATE" "UPDATETEXT"
 "UPPER" "UPPERCASE" "USAGE" "USE" "USER" "USING" "UTC_DATE"
"UTC_TIME" "UTC_TIMESTAMP" "VALIDATE" "VALIDPROC" "VALUE" "VALUES" "VALUE_OF"
 "VARBINARY" "VARBYTE" "VARCHAR" "VARCHAR2" "VARCHARACTER" "VARGRAPHIC"
"VARIABLE" "VARIADIC" "VARIANT" "VARYING" "VAR_POP" "VAR_SAMP" "VCAT" "VERBOSE"
 "VERSIONING" "VIEW" "VIRTUAL" "VOLATILE" "VOLUMES" "WAIT" "WAITFOR"
"WHEN" "WHENEVER" "WHERE" "WHILE" "WIDTH_BUCKET" "WINDOW" "WITH" "WITHIN"
 "WITHIN_GROUP" "WITHOUT" "WLM" "WORK" "WRITE" "WRITETEXT" "XMLCAST" "XMLEXISTS"
"XMLNAMESPACES" "XOR" "YEAR" "YEARS" "YEAR_MONTH" "ZEROFILL" "ZEROIFNULL" "ZONE"
```

Alarms on CloudWatch Metrics Insights queries in CloudWatch

You can create alarms on Metrics Insights queries. This helps you have alarms that track multiple resources without needing to be updated later. The query catches new resources and resources that change. For example, you can create an alarm that watches the CPU utilization of your fleet, and the alarm automatically evaluates new instances that you launch after creating the alarm.

In a monitoring account that is set up for CloudWatch cross-account observability, your Metrics Insights alarms can watch resources in source accounts and in the monitoring account itself. For more information about how to limit your alarm queries to a specific account or to group the results by account ID, see the `WHERE` and `GROUP BY` sections in Query components and syntax in CloudWatch Metrics Insights.

Contents

Creating a Metrics Insights CloudWatch alarm

To create an alarm on a Metrics Insights query using the console

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Metrics**, **All metrics**.
3. (Optional) To run a pre-built sample query, choose **Add query** and select the query to run. Or, you can choose **Editor** to edit the sample query and then choose **Run** to run the modified query.
4. To create your own query, choose **Multi source query**. You can then use the **Builder** view, the **Editor** view, and also use a combination of both. You can switch between the two views anytime and see your work in progress in both views.

   In the **Builder** view, you can browse and select the metric namespace, metric name, filter, group, and order options. For each of these options, the query builder offers you a list of possible selections from your environment to choose from.

   In the **Editor** view, you can start writing your query. As you type, the editor offers suggestions based on the characters that you have typed so far.

   > **Important**
   >
   > To set an alarm on a Metrics Insights query, the query must return a single time series. If it contains a GROUP BY statement, the GROUP BY statement must be wrapped inside a metric math expression that returns only one time series as the final result of the expression.

5. When you are satisfied with your query, choose **Run**.
6. Choose **Create alarm**.
7. Under **Conditions**, specify the following:

   a. For **Whenever *`metric`* is**, specify whether the metric must be greater than, less than, or equal to the threshold. Under **than...**, specify the threshold value.

   b. Choose **Additional configuration**. For **Datapoints to alarm**, specify how many evaluation periods (data points) must be in the `ALARM` state to trigger the alarm. If the two values here match, you create an alarm that goes to `ALARM` state if that many consecutive periods are breaching.

To create an M out of N alarm, specify a lower number for the first value than you specify for the second value. For more information, see Evaluating an alarm.

c. For **Missing data treatment**, choose how to have the alarm behave when some data points are missing. For more information, see Configuring how CloudWatch alarms treat missing data.

8. Choose **Next**.
9. Under **Notification**, select an SNS topic to notify when the alarm is in `ALARM` state, `OK` state, or `INSUFFICIENT_DATA` state.

   To have the alarm send multiple notifications for the same alarm state or for different alarm states, choose **Add notification**.

   To have the alarm not send notifications, choose **Remove**.
10. To have the alarm perform Auto Scaling, EC2, or Systems Manager actions, choose the appropriate button and choose the alarm state and action to perform. Alarms can perform Systems Manager actions only when they go into ALARM state. For more information about Systems Manager actions, see Configuring CloudWatch to create OpsItems from alarms and Incident creation.

    Note

    To create an alarm that performs an SSM Incident Manager action, you must have certain permissions. For more information, see Identity-based policy examples for Amazon Systems Manager Incident Manager.
11. When finished, choose **Next**.
12. Enter a name and description for the alarm. The name must contain only ASCII characters. Then choose **Next**.
13. Under **Preview and create**, confirm that the information and conditions are what you want, then choose **Create alarm**.

To create an alarm on a Metrics Insights query using the Amazon CLI

- Use the `put-metric-alarm` command and specify a Metrics Insights query in the `metrics` parameter. For example, the following command sets an alarm that goes into ALARM state if any of your instances go above 50% in CPU utilization.

```
aws cloudwatch put-metric-alarm --alarm-name Metrics-Insights-alarm --
evaluation-periods 1 --comparison-operator GreaterThanThreshold --metrics
 '[{"Id":"m1","Expression":"SELECT MAX(CPUUtilization) FROM SCHEMA(\"AWS/EC2\",
 InstanceId)", "Period":60}]' --threshold 50
```

How partial data from a Metrics Insights query is evaluated

If the Metrics Insights query used for the alarm matches more than 10,000 metrics, the alarm is evaluated based on the first 10,000 metrics that the query finds. This means that the alarm is being evaluated on partial data.

You can use the following methods to find whether a Metrics Insights alarm is currently evaluating its alarm state based on partial data:

- In the console, if you choose an alarm to see the **Details** page, the message **Evaluation warning: Not evaluating all data** appears on that page.
- You see the value `PARTIAL_DATA` in the `EvaluationState` field when you use the describe-alarms Amazon CLI command or the DescribeAlarms API.

Alarms also publish events to Amazon EventBridge when it goes into the partial data state, so you can create an EventBridge rule to watch for these events. In these events, the `evaluationState` field has the value `PARTIAL_DATA`. The following is an example.

```
{
    "version": "0",
    "id": "12345678-3bf9-6a09-dc46-12345EXAMPLE",
    "detail-type": "CloudWatch Alarm State Change",
    "source": "aws.cloudwatch",
    "account": "123456789012",
    "time": "2022-11-08T11:26:05Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:cloudwatch:us-east-1:123456789012:alarm:my-alarm-name"
    ],
    "detail": {
        "alarmName": "my-alarm-name",
        "state": {
            "value": "ALARM",
```

```
            "reason": "Threshold Crossed: 3 out of the last 3 datapoints [20000.0
 (08/11/22 11:25:00), 20000.0 (08/11/22 11:24:00), 20000.0 (08/11/22 11:23:00)] were
 greater than the threshold (0.0) (minimum 1 datapoint for OK -> ALARM transition).",
            "reasonData": "{\"version\":\"1.0\",\"queryDate\":
\"2022-11-08T11:26:05.399+0000\",\"startDate\":\"2022-11-08T11:23:00.000+0000\",
\"period\":60,\"recentDatapoints\":[20000.0,20000.0,20000.0],\"threshold\":0.0,
\"evaluatedDatapoints\":[{\"timestamp\":\"2022-11-08T11:25:00.000+0000\",\"value
\":20000.0}]}",
            "timestamp": "2022-11-08T11:26:05.401+0000",
            "evaluationState": "PARTIAL_DATA"
        },
        "previousState": {
            "value": "INSUFFICIENT_DATA",
            "reason": "Unchecked: Initial alarm creation",
            "timestamp": "2022-11-08T11:25:51.227+0000"
        },
        "configuration": {
            "metrics": [
                {
                    "id": "m2",
                    "expression": "SELECT SUM(PartialDataTestMetric) FROM
 partial_data_test",
                    "returnData": true,
                    "period": 60
                }
            ]
        }
    }
}
```

If the query for the alarm includes a GROUP BY statement that initially returns more than 500 time series, the alarm is evaluated based on the first 500 time series that the query finds. However, if you use an ORDER BY clause, then all the time series that the query finds are sorted, and the 500 that have the highest or lowest values according to your ORDER BY clause are used to evaluate the alarm.

Use Metrics Insights queries with metric math

You can use a Metrics Insights query as an input to a metric math function. For more information about metric math, see Using math expressions with CloudWatch metrics.

A Metrics Insights query that does not include a **GROUP BY** clause returns a single time series. Therefore, its returned results can be used with any metric math function that takes a single time series as input.

A Metrics Insights query that includes a **GROUP BY** clause returns multiple time series. Therefore, its returned results can be used with any metric math function that takes an array of time series as input.

For example, the following query returns the total number of bytes downloaded for each bucket in the Region, as an array of time series:

```
SELECT SUM(BytesDownloaded)
FROM SCHEMA("AWS/S3", BucketName, FilterId)
WHERE FilterId = 'EntireBucket'
GROUP BY BucketName
```

On a graph in the console or in a GetMetricData operation, the results of this query are q1. This query returns the result in bytes, so if you want to see the result as MB instead, you can use the following math function:

```
q1/1024/1024
```

Use natural language to generate and update CloudWatch Metrics Insights queries

CloudWatch supports a natural language query capability to help you generate and update queries for CloudWatch Metrics Insights and CloudWatch Logs Insights.

With this capability, you can ask questions about or describe the CloudWatch data you're looking for in plain English. The natural language capability generates a query based on a prompt that you enter and provides a line-by-line explanation of how the query works. You can also update your query to further investigate your data.

Depending on your environment, you can enter prompts like "Which Amazon Elastic Compute Cloud instance has the highest network out?" and "Show me the top 10 Amazon DynamoDB Tables by consumed reads."

Note

The natural-language query feature is generally available in 10 Regions. For some Regions, the feature makes cross-Region calls to Regions in the United States to process the query prompts. The following table lists the supported Regions, and shows where each Region processes its prompts.

Supported Region	Region where prompt is processed
US East (N. Virginia)	US East (N. Virginia)
US East (Ohio)	US East (N. Virginia)
US West (Oregon)	US West (Oregon)
Asia Pacific (Hong Kong)	US West (Oregon)
Asia Pacific (Singapore)	US West (Oregon)
Asia Pacific (Sydney)	US West (Oregon)
Asia Pacific (Tokyo)	Asia Pacific (Tokyo)
Europe (Frankfurt)	Europe (Frankfurt)
Europe (Ireland)	US East (N. Virginia)
Europe (Stockholm)	US East (N. Virginia)

To generate a CloudWatch Metrics Insights query with this capability, open the CloudWatch Metrics Insights query editor in the *builder* or *editor* view and choose **Generate query**.

Important

To use the natural language query capability, you must use the CloudWatchFullAccess, CloudWatchReadOnlyAccess, CloudWatchFullAccessV2, AdministratorAccess, or ReadOnlyAccess policy.

You can also include the `cloudwatch:GenerateQuery` action in a new or existing customer managed or inline policy.

Example queries

The examples in this section describe how to generate and update queries using the natural language capability.

Note

For more information on the CloudWatch Metrics Insights query editor and syntax, see CloudWatch Metrics Insights query components and syntax.

Example: Generate a natural language query

To generate a query using natural language, enter a prompt and choose **Generate new query**. This example shows a query that performs a basic search.

Prompt

The following is an example of a prompt that directs the capability to search for the top 10 DynamoDB Tables that consume the most read capacity.

```
Show top 10 DynamoDB Tables by consumed reads
```

Query

The following is an example of a query that the natural language capability generates based on the prompt. Notice how the prompt appears in a comment before the query. After the query, you can read an explanation that describes how the query works.

```
# Show top 10 DynamoDB Tables by consumed reads
SELECT SUM("ConsumedReadCapacityUnits")
FROM "AWS/DynamoDB"
GROUP BY TableName
ORDER BY SUM() DESC
LIMIT 10
```

```
# This query selects the sum of consumed read capacity units for each DynamoDB table,
 groups the results by table name, orders the results from highest to lowest read
 capacity consumption, and limits the results to the top 10 tables.
```

> **Note**
>
> To turn off the appearance of your prompt and the explanation of how the query works, use the gear icon in your editor.

Example: Update a natural language query

You can update a query by editing the initial prompt and then choosing **Update query**.

Updated prompt

The following example shows an updated version of the previous prompt. Instead of a prompt that searches for the top 10 DynamoDB Tables that consume the most read capacity, this prompt now directs the capability to sort the results by the number of bytes returned.

```
Sort by bytes returned instead
```

Updated query

The following is an example of the updated query. Notice how the updated prompt appears in a comment before the updated query. After the query, you can read an explanation that describes how the original query has been updated.

```
# Sort by bytes returned instead
SELECT SUM("ReturnedBytes")
FROM "AWS/DynamoDB"
GROUP BY TableName
ORDER BY SUM() DESC
LIMIT 10
# This query modifies the original query to select the sum of returned bytes instead
 of consumed read capacity units, and orders the results from highest to lowest sum of
 returned bytes, limiting the results to the top 10 tables.
```

Opting out of using your data for service improvement

The natural language prompt data you provide to train the AI model and generate relevant queries is used solely to provide and maintain your service. This data might be used to improve the quality of CloudWatch Metrics Insights. Your trust and privacy, as well as the security of your content, is our highest priority. For more information, see Amazon Service Terms and Amazon responsible AI policy.

You can opt out of having your content used to develop or improve the quality of natural language queries by creating an AI service opt-out policy. To opt-out of data collection for all CloudWatch AI features, including the query generation capability, you must create an opt-out policy for CloudWatch. For more information, see AI services opt-out policies in the *Amazon Organizations User Guide*.

SQL inference

CloudWatch Metrics Insights uses several mechanisms to infer the intention of a given SQL query.

Topics

- Time bucketing
- Fields projection
- ORDER BY global aggregation

Time bucketing

Time series data points resulting from a query are rolled up into time buckets based on the requested period. To aggregate values in standard SQL, an explicit GROUP BY clause must be defined to collect all the observations of a given period together. Because this is the standard way to query time series data, CloudWatch Metrics Insights infers time bucketing without the need to express an explicit **GROUP BY** clause.

For example, when a query is performed with a period of one minute, all the observations belonging to that minute until the next (excluded) are rolled up to the start time of the time bucket. This makes Metrics Insights SQL statements more concise and less verbose.

```
SELECT AVG(CPUUtilization)
FROM SCHEMA("AWS/EC2", InstanceId)
```

The previous query returns a single time series (timestamp-value pairs), representing the average CPU utilization of all Amazon EC2 instances. Assuming the requested period is one minute, each data point returned represents the average of all observations measured within a specific one-minute interval (start time inclusive, end time exclusive). The timestamp related to the specific data point is the start time of the bucket

Fields projection

Metrics Insights queries always return the timestamp projection. You don't need to specify a timestamp column in the **SELECT** clause to get the timestamp of each corresponding data point value. For details about how timestamp is calculated, see Time bucketing.

When using **GROUP BY**, each group name is also inferred and projected in the result, so that you can group the returned time series.

```
SELECT AVG(CPUUtilization)
FROM SCHEMA("AWS/EC2", InstanceId)
GROUP BY InstanceId
```

The previous query returns a time series for each Amazon EC2 instance. Each time series is labeled after the value of the instance ID.

ORDER BY global aggregation

When using **ORDER BY**, **FUNCTION()** infers which aggregate function that you want to order by (the data point values of the queried metrics). The aggregate operation is performed across all the matched data points of each individual time series across the queried time window.

```
SELECT AVG(CPUUtilization)
FROM SCHEMA("AWS/EC2", InstanceId)
GROUP BY InstanceId
ORDER BY MAX()
LIMIT 10
```

The previous query returns the CPU utilization for each Amazon EC2 instance, limiting the result set to 10 entries. The results are ordered based on the maximum value of the individual time series within the requested time window. The **ORDER BY** clause is applied before **LIMIT**, so that the ordering is calculated against more than 10 time series.

Metrics Insights sample queries

This section contains examples of useful CloudWatch Metrics Insights queries that you can copy and use directly or copy and modify in query editor. Some of these examples are already available in the console, and you can access them by choosing **Add query** in the **Metrics** view.

Application Load Balancer examples

Total requests across all load balancers

```
SELECT SUM(RequestCount)
FROM SCHEMA("AWS/ApplicationELB", LoadBalancer)
```

Top 10 most active load balancers

```
SELECT MAX(ActiveConnectionCount)
FROM SCHEMA("AWS/ApplicationELB", LoadBalancer)
GROUP BY LoadBalancer
ORDER BY SUM() DESC
LIMIT 10
```

Amazon API usage examples

Top 20 Amazon APIs by the number of calls in your account

```
SELECT COUNT(CallCount)
FROM SCHEMA("AWS/Usage", Class, Resource, Service, Type)
WHERE Type = 'API'
GROUP BY Service, Resource
ORDER BY COUNT() DESC
LIMIT 20
```

CloudWatch APIs sorted by calls

```
SELECT COUNT(CallCount)
FROM SCHEMA("AWS/Usage", Class, Resource, Service, Type)
WHERE Type = 'API' AND Service = 'CloudWatch'
GROUP BY Resource
ORDER BY COUNT() DESC
```

DynamoDB examples

Top 10 tables by consumed reads

```
SELECT SUM(ProvisionedWriteCapacityUnits)
FROM SCHEMA("AWS/DynamoDB", TableName)
GROUP BY TableName
ORDER BY MAX() DESC LIMIT 10
```

Top 10 tables by returned bytes

```
SELECT SUM(ReturnedBytes)
FROM SCHEMA("AWS/DynamoDB", TableName)
GROUP BY TableName
ORDER BY MAX() DESC LIMIT 10
```

Top 10 tables by user errors

```
SELECT SUM(UserErrors)
FROM SCHEMA("AWS/DynamoDB", TableName)
GROUP BY TableName
ORDER BY MAX() DESC LIMIT 10
```

Amazon Elastic Block Store examples

Top 10 Amazon EBS volumes by bytes written

```
SELECT SUM(VolumeWriteBytes)
FROM SCHEMA("AWS/EBS", VolumeId)
GROUP BY VolumeId
ORDER BY SUM() DESC
LIMIT 10
```

Average Amazon EBS volume write time

```
SELECT AVG(VolumeTotalWriteTime)
FROM SCHEMA("AWS/EBS", VolumeId)
```

Amazon EC2 examples

CPU utilization of EC2 instances sorted by highest

```
SELECT AVG(CPUUtilization)
  FROM SCHEMA("AWS/EC2", InstanceId)
  GROUP BY InstanceId
  ORDER BY AVG() DESC
```

Average CPU utilization across the entire fleet

```
SELECT AVG(CPUUtilization)
FROM SCHEMA("AWS/EC2", InstanceId)
```

Top 10 instances by highest CPU utilization

```
SELECT MAX(CPUUtilization)
FROM SCHEMA("AWS/EC2", InstanceId)
GROUP BY InstanceId
ORDER BY MAX() DESC
LIMIT 10
```

In this case, the CloudWatch agent is collecting a `CPUUtilization` metric per application. This query filters the average of this metric for a specific application name.

```
SELECT AVG(CPUUtilization)
FROM "AWS/CWAgent"
WHERE ApplicationName = 'eCommerce'
```

Amazon Elastic Container Service examples

Average CPU utilization across all ECS clusters

```
SELECT AVG(CPUUtilization)
FROM SCHEMA("AWS/ECS", ClusterName)
```

Top 10 clusters by memory utilization

```
SELECT AVG(MemoryUtilization)
FROM SCHEMA("AWS/ECS", ClusterName)
GROUP BY ClusterName
ORDER BY AVG() DESC
LIMIT 10
```

Top 10 services by CPU utilization

```
SELECT AVG(CPUUtilization)
FROM SCHEMA("AWS/ECS", ClusterName, ServiceName)
GROUP BY ClusterName, ServiceName
ORDER BY AVG() DESC
LIMIT 10
```

Top 10 services by running tasks (Container Insights)

```
SELECT AVG(RunningTaskCount)
FROM SCHEMA("ECS/ContainerInsights", ClusterName, ServiceName)
GROUP BY ClusterName, ServiceName
ORDER BY AVG() DESC
LIMIT 10
```

Amazon Elastic Kubernetes Service Container Insights examples

Average CPU utilization across all EKS clusters

```
SELECT AVG(pod_cpu_utilization)
FROM SCHEMA("ContainerInsights", ClusterName)
```

Top 10 clusters by node CPU utilization

```
SELECT AVG(node_cpu_utilization)
FROM SCHEMA("ContainerInsights", ClusterName)
GROUP BY ClusterName
ORDER BY AVG() DESC LIMIT 10
```

Top 10 clusters by pod memory utilization

```
SELECT AVG(pop_memory_utilization)
FROM SCHEMA("ContainerInsights", ClusterName)
GROUP BY ClusterName
ORDER BY AVG() DESC LIMIT 10
```

Top 10 nodes by CPU utilization

```
SELECT AVG(node_cpu_utilization)
```

```
FROM SCHEMA("ContainerInsights", ClusterName, NodeName)
GROUP BY ClusterName, NodeName
ORDER BY AVG() DESC LIMIT 10
```

Top 10 pods by memory utilization

```
SELECT AVG(pod_memory_utilization)
FROM SCHEMA("ContainerInsights", ClusterName, PodName)
GROUP BY ClusterName, PodName
ORDER BY AVG() DESC LIMIT 10
```

EventBridge examples

Top 10 rules by invocations

```
SELECT SUM(Invocations)
FROM SCHEMA("AWS/Events", RuleName)
GROUP BY RuleName
ORDER BY MAX() DESC LIMIT 10
```

Top 10 rules by failed invocations

```
SELECT SUM(FailedInvocations)
FROM SCHEMA("AWS/Events", RuleName)
GROUP BY RuleName
ORDER BY MAX() DESC LIMIT 10
```

Top 10 rules by matched rules

```
SELECT SUM(MatchedEvents)
FROM SCHEMA("AWS/Events", RuleName)
GROUP BY RuleName
ORDER BY MAX() DESC LIMIT 10
```

Kinesis examples

Top 10 streams by bytes written

```
SELECT SUM("PutRecords.Bytes")
```

```
FROM SCHEMA("AWS/Kinesis", StreamName)
GROUP BY StreamName
ORDER BY SUM() DESC LIMIT 10
```

Top 10 streams by earliest items in the stream

```
SELECT MAX("GetRecords.IteratorAgeMilliseconds")
FROM SCHEMA("AWS/Kinesis", StreamName)
GROUP BY StreamName
ORDER BY MAX() DESC LIMIT 10
```

Lambda examples

Lambda functions ordered by number of invocations

```
SELECT SUM(Invocations)
FROM SCHEMA("AWS/Lambda", FunctionName)
GROUP BY FunctionName
ORDER BY SUM() DESC
```

Top 10 Lambda functions by longest runtime

```
SELECT AVG(Duration)
FROM SCHEMA("AWS/Lambda", FunctionName)
GROUP BY FunctionName
ORDER BY MAX() DESC
LIMIT 10
```

Top 10 Lambda functions by error count

```
SELECT SUM(Errors)
FROM SCHEMA("AWS/Lambda", FunctionName)
GROUP BY FunctionName
ORDER BY SUM() DESC
LIMIT 10
```

CloudWatch Logs examples

Top 10 log groups by incoming events

```
SELECT SUM(IncomingLogEvents)
FROM SCHEMA("AWS/Logs", LogGroupName)
GROUP BY LogGroupName
ORDER BY SUM() DESC LIMIT 10
```

Top 10 log groups by written bytes

```
SELECT SUM(IncomingBytes)
FROM SCHEMA("AWS/Logs", LogGroupName)
GROUP BY LogGroupName
ORDER BY SUM() DESC LIMIT 10
```

Amazon RDS examples

Top 10 Amazon RDS instances by highest CPU utilization

```
SELECT MAX(CPUUtilization)
FROM SCHEMA("AWS/RDS", DBInstanceIdentifier)
GROUP BY DBInstanceIdentifier
ORDER BY MAX() DESC
LIMIT 10
```

Top 10 Amazon RDS clusters by writes

```
SELECT SUM(WriteIOPS)
FROM SCHEMA("AWS/RDS", DBClusterIdentifier)
GROUP BY DBClusterIdentifier
ORDER BY MAX() DESC
LIMIT 10
```

Amazon Simple Storage Service examples

Average latency by bucket

```
SELECT AVG(TotalRequestLatency)
FROM SCHEMA("AWS/S3", BucketName, FilterId)
WHERE FilterId = 'EntireBucket'
GROUP BY BucketName
ORDER BY AVG() DESC
```

Top 10 buckets by bytes downloaded

```
SELECT SUM(BytesDownloaded)
FROM SCHEMA("AWS/S3", BucketName, FilterId)
WHERE FilterId = 'EntireBucket'
GROUP BY BucketName
ORDER BY SUM() DESC
LIMIT 10
```

Amazon Simple Notification Service examples

Total messages published by SNS topics

```
SELECT SUM(NumberOfMessagesPublished)
FROM SCHEMA("AWS/SNS", TopicName)
```

Top 10 topics by messages published

```
SELECT SUM(NumberOfMessagesPublished)
FROM SCHEMA("AWS/SNS", TopicName)
GROUP BY TopicName
ORDER BY SUM() DESC
LIMIT 10
```

Top 10 topics by message delivery failures

```
SELECT SUM(NumberOfNotificationsFailed)
FROM SCHEMA("AWS/SNS", TopicName)
GROUP BY TopicName
ORDER BY SUM() DESC
LIMIT 10
```

Amazon SQS examples

Top 10 queues by number of visible messages

```
SELECT AVG(ApproximateNumberOfMessagesVisible)
FROM SCHEMA("AWS/SQS", QueueName)
GROUP BY QueueName
```

```
ORDER BY AVG() DESC
LIMIT 10
```

Top 10 most active queues

```
SELECT SUM(NumberOfMessagesSent)
FROM SCHEMA("AWS/SQS", QueueName)
GROUP BY QueueName
ORDER BY SUM() DESC
LIMIT 10
```

Top 10 queues by age of earliest message

```
SELECT AVG(ApproximateAgeOfOldestMessage)
FROM SCHEMA("AWS/SQS", QueueName)
GROUP BY QueueName
ORDER BY AVG() DESC
LIMIT 10
```

Metrics Insights limits

CloudWatch Metrics Insights currently has the following limits:

- Currently, you can query only the most recent three hours of data.
- A single query can process no more than 10,000 metrics. This means that if the **SELECT**, **FROM**, and **WHERE** clauses match more than 10,000 metrics, the query only processes the first 10,000 of these metrics that it finds.
- A single query can return no more than 500 time series. This means that if the query would return more than 500 metrics, not all metrics will be returned in the query results. If you use an **ORDER BY** clause, then all the metrics being processed are sorted, and the 500 that have the highest or lowest values according to your **ORDER BY** clause are returned.

  If you do not include an **ORDER BY** clause, you can't control which 500 matching metrics are returned.
- You can have as many as 200 Metrics Insights alarms per Region.
- Metrics Insights does not support high-resolution data, which is metric data reported with a granularity of less than one minute. If you request high-resolution data, the request does not fail, but the output is aggregated at one-minute granularity.

- Each GetMetricData operation can have only one query, but you can have multiple widgets in a dashboard that each include a query.

Metrics Insights glossary

label

In Metrics Insights, a label is a key-value pair that is used to scope a query to return a particular set of data, or to define criteria by which query results are to be separated into separate time series. A label key is similar to a column name in SQL. Currently, labels must be CloudWatch metric dimensions.

observation

An observation is a value recorded for a given metric at a given time.

Troubleshooting Metrics Insights

The results include "Other," but I don't have this as a dimension

This means that the query includes a **GROUP BY** clause that specifies a label key that is not used in some of the metrics that are returned by the query. In this case, a null group named `Other` is returned. The metrics that do not include that label key are probably aggregated metrics that return values aggregated across all values of that label key.

For example, suppose we have the following query:

```
SELECT AVG(Faults)
FROM MyCustomNamespace
GROUP BY Operation, ServiceName
```

If some of the returned metrics don't include `ServiceName` as a dimension, then those metrics are displayed as having `Other` as the value for `ServiceName`.

To prevent seeing "Other" in your results, use **SCHEMA** in your **FROM** clause, as in the following example:

```
SELECT AVG(Faults)
FROM SCHEMA(MyCustomNamespace, Operation)
```

```
GROUP BY Operation, ServiceName
```

This limits the returned results to only the metrics that have both the `Operation` and `ServiceName` dimensions.

The oldest timestamp in my graph has a lower metric value than the others

CloudWatch Metrics Insights currently supports the latest three hours of data only. When you graph with a period larger than one minute, there could be cases where the oldest data point differs from the expected value. This is because the Metrics Insights queries return only the most recent three hours of data. In this case, the oldest data point in the query returns only the observations that have been measured within the last three hours boundary, instead of returning all the observations within the period of that data point.

Use metrics explorer to monitor resources by their tags and properties

Metrics explorer is a tag-based tool that enables you to filter, aggregate, and visualize your metrics by tags and resource properties, to enhance observability for your services. This gives you a flexible and dynamic troubleshooting experience, so that you can create multiple graphs at a time and use these graphs to build your application health dashboards.

Metrics explorer visualizations are dynamic, so if a matching resource is created after you create a metrics explorer widget and add it to a CloudWatch dashboard, the new resource automatically appears in the explorer widget.

For example, if all of your EC2 production instances have the **`production`** tag, you can use metrics explorer to filter and aggregate metrics from all of these instances to understand their health and performance. If a new instance with a matching tag is later created, it's automatically added to the metrics explorer widget.

> **Note**
>
> Metrics explorer provides a point-in-time experience. Resources that have been terminated, or no longer exist with the property or tag that you specified are not displayed in the visualisation. However, you can still find the metrics for these resources in CloudWatch metrics views.

With metrics explorer, you can choose how to aggregate metrics from the resources that match the criteria, and whether to show them all in a single graph or on different graphs within one metrics explorer widget.

Metrics explorer includes templates that you can use to see useful visualization graphs with one click, and you can also extend these templates to create completely customized metrics explorer widgets.

You can use metrics explorer across accounts if you are using the Cross-account cross-Region CloudWatch console. However, metrics explorer is not supported for cross-account use from a monitoring account in CloudWatch cross-account observability.

Metrics explorer supports metrics emitted by Amazon and EC2 metrics that are published by the CloudWatch agent, including memory, disk, and CPU metrics. To use metrics explorer to see the metrics that are published by the CloudWatch agent, you might have to update your CloudWatch agent configuration file. For more information, see CloudWatch agent configuration for metrics explorer

To create a visualization with metrics explorer and optionally add it to a dashboard, follow these steps.

To create a visualization with metrics explorer

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Explorer**.
3. Do one of the following:
   - To use a template, select it in the box that currently shows **Empty Explorer**.

     Depending on the template, the explorer might immediately display graphs of metrics. If it doesn't, choose one or more tags or properties in the **From** box and then data should appear. If it doesn't, use the options at the top of the page to display a longer time range in the graphs.
   - To create a custom visualization, under **Metrics**, choose a single metric or all the available metrics from a service.

     After you choose a metric, you can optionally repeat this step to add more metrics.
4. For each metric selected, CloudWatch displays the statistic that it will use immediately after the metric name. To change this, choose the statistic name, and then choose the statistic that you want.

5. Under **From**, choose a tag or a resource property to filter your results.

   After you do this, you can optionally repeat this step to choose more tags or resource properties.

   If you choose multiple values of the same property, such as two EC2 instance types, the explorer displays all the resources that match either chosen property. It's treated as an OR operation.

   If you choose different properties or tags, such as the **`Production`** tag and the M5 instance type, only the resources that match all of these selections are displayed. It's treated as an AND operation.
6. (Optional) For **Aggregate by**, choose a statistic to use to aggregate the metrics. Then, next to **for**, choose how to aggregate the metric from the list. You can aggregate together all the resources that are currently displayed, or aggregate by a single tag or resource property.

   Depending on how you choose to aggregate, the result may be a single time series or multiple time series.
7. Under **Split by**, you can choose to split a single graph with multiple time series into multiple graphs. The split can be made by a variety of criteria, which you choose under **Split by**.
8. Under **Graph options**, you can refine the graph by changing the period, the type of graph, the legend placement, and the layout.
9. To add this visualization as a widget to a CloudWatch dashboard, choose **Add to dashboard**.

CloudWatch agent configuration for metrics explorer

To enable metrics explorer to discover EC2 metrics published by the CloudWatch agent, make sure that the CloudWatch agent configuration file contains the following values:

- In the `metrics` section, make sure that the `aggregation_dimensions` parameter includes `["InstanceId"]`. It can also contain other dimensions.
- In the `metrics` section, make sure that the `append_dimensions` parameter includes a `{"InstanceId":"${aws:InstanceId}"}` line. It can also contain other lines.
- In the `metrics` section, inside the `metrics_collected` section, check the sections for each resource type that you want metrics explorer to discover, such as the `cpu`, `disk`, and `memory` sections. Make sure that each of these sections has a `"resources": [ "*"] line.`.

- In the `cpu` section of the `metrics_collected` section, make sure there is a `"totalcpu": true` line.
- You must use the default `CWAgent` namespace for the metrics collected by the CloudWatch agent, instead of a custom namespace.

The settings in the previous list cause the CloudWatch agent to publish aggregate metrics for disks, CPUs, and other resources that can be plotted in metrics explorer for all the instances that use it.

These settings will republish the metrics that you had previously set up to be published with multiple dimensions, adding to your metric costs.

For more information about editing the CloudWatch agent configuration file, see Manually create or edit the CloudWatch agent configuration file.

Use metric streams

You can use *metric streams* to continually stream CloudWatch metrics to a destination of your choice, with near-real-time delivery and low latency. Supported destinations include Amazon destinations such as Amazon Simple Storage Service and several third-party service provider destinations.

There are three main usage scenarios for CloudWatch metric streams:

- **Custom setup with Firehose**— Create a metric stream and direct it to an Amazon Data Firehose delivery stream that delivers your CloudWatch metrics to where you want them to go. You can stream them to a data lake such as Amazon S3, or to any destination or endpoint supported by Firehose including third-party providers. JSON, OpenTelemetry 1.0.0, and OpenTelemetry 0.7.0 formats are supported natively, or you can configure transformations in your Firehose delivery stream to convert the data to a different format such as Parquet. With a metric stream, you can continually update monitoring data, or combine this CloudWatch metric data with billing and performance data to create rich datasets. You can then use tools such as Amazon Athena to get insight into cost optimization, resource performance, and resource utilization.
- **Quick S3 setup**— Stream to Amazon Simple Storage Service with a quick setup process. By default, CloudWatch creates the resources needed for the stream. JSON, OpenTelemetry 1.0.0, and OpenTelemetry 0.7.0 formats are supported.
- **Quick Amazon partner setup**— CloudWatch provides a quick setup experience for some third-party partners. You can use third-party service providers to monitor, troubleshoot, and analyze

your applications using the streamed CloudWatch data. When you use the quick partner setup workflow, you need to provide only a destination URL and API key for your destination, and CloudWatch handles the rest of the setup. Quick partner setup is available for the following third-party providers:

- Datadog
- Dynatrace
- Elastic
- New Relic
- Splunk Observability Cloud
- SumoLogic

You can stream all of your CloudWatch metrics, or use filters to stream only specified metrics. Each metric stream can include up to 1000 filters that either include or exclude metric namespaces or specific metrics. A single metric stream can have only include or exclude filters, but not both.

After a metric stream is created, if new metrics are created that match the filters in place, the new metrics are automatically included in the stream.

There is no limit on the number of metric streams per account or per Region, and no limit on the number of metric updates being streamed.

Each stream can use either JSON format, OpenTelemetry 1.0.0, or OpenTelemetry 0.7.0 format. You can edit the output format of a metric stream at any time, such as for upgrading from OpenTelemetry 0.7.0 to OpenTelemetry 1.0.0. For more information about output formats, see CloudWatch metric stream output in JSON format, CloudWatch metric stream output in OpenTelemetry 1.0.0 format, and CloudWatch metric stream output in OpenTelemetry 0.7.0 format.

For metric streams in monitoring accounts, you can choose whether to include metrics from the source accounts linked to that monitoring account. For more information, see CloudWatch cross-account observability.

Metric streams always include the `Minimum`, `Maximum`, `SampleCount`, and `Sum` statistics. You can also choose to include additional statistics at an additional charge. For more information, see Statistics that can be streamed.

Metric streams pricing is based on the number of metric updates. You also incur charges from Firehose for the delivery stream used for the metric stream. For more information, see Amazon CloudWatch Pricing.

Topics

Set up a metric stream

Use the steps in the following sections to set up a CloudWatch metric stream.

After a metric stream is created, the time it takes for metric data to appear at the destination depends on the configured buffering settings on the Firehose delivery stream. The buffering is expressed in maximum payload size or maximum wait time, whichever is reached first. If these are set to the minimum values (60 seconds, 1MB) the expected latency is within 3 minutes if the selected CloudWatch namespaces have active metric updates.

In a CloudWatch metric stream, data is sent every minute. Data might arrive at the final destination out of order. All specified metrics in the specified namespaces are sent in the metric stream, except metrics with a timestamp that is more than two days old.

For each combination of metric name and namespace that you stream, all dimension combinations of that metric name and namespace are streamed.

For metric streams in monitoring accounts, you can choose whether to include metrics from the source accounts linked to that monitoring account. For more information, see CloudWatch cross-account observability.

To create and manage metric streams, you must be logged on to an account that has the **CloudWatchFullAccess** policy and the `iam:PassRole` permission, or an account that has the following list of permissions:

- `iam:PassRole`
- `cloudwatch:PutMetricStream`
- `cloudwatch:DeleteMetricStream`
- `cloudwatch:GetMetricStream`
- `cloudwatch:ListMetricStreams`
- `cloudwatch:StartMetricStreams`
- `cloudwatch:StopMetricStreams`

If you're going to have CloudWatch set up the IAM role needed for metric streams, you must also have the `iam:CreateRole` and `iam:PutRolePolicy` permissions.

> **Important**
>
> A user with the `cloudwatch:PutMetricStream` has access to the CloudWatch metric data that is being streamed, even if they don't have the `cloudwatch:GetMetricData` permission.

Topics

- Custom setup with Firehose
- Use Quick Amazon S3 setup
- Quick partner setup

Custom setup with Firehose

Use this method to create a metric stream and direct it to an Amazon Data Firehose delivery stream that delivers your CloudWatch metrics to where you want them to go. You can stream them to a data lake such as Amazon S3, or to any destination or endpoint supported by Firehose including third-party providers.

JSON, OpenTelemetry 1.0.0, and OpenTelemetry 0.7.0 formats are supported natively, or you can configure transformations in your Firehose delivery stream to convert the data to a different format such as Parquet. With a metric stream, you can continually update monitoring data, or combine this CloudWatch metric data with billing and performance data to create rich datasets. You can then use tools such as Amazon Athena to get insight into cost optimization, resource performance, and resource utilization.

You can use the CloudWatch console, the Amazon CLI, Amazon CloudFormation, or the Amazon Cloud Development Kit (Amazon CDK) to set up a metric stream.

The Firehose delivery stream that you use for your metric stream must be in the same account and the same Region where you set up the metric stream. To achieve cross-Region functionality, you can configure the Firehose delivery stream to stream to a final destination that is in a different account or a different Region.

CloudWatch console

This section describes how to use the CloudWatch console to set up a metric stream using Firehose.

To set up a custom metric stream using Firehose

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Metrics**, **Streams**. Then choose **Create metric stream**.
3. (Optional) If you are signed in to an account that is set up as a monitoring account in CloudWatch cross-account observability, you can choose whether to include metrics from linked source accounts in this metric stream. To include metrics from source accounts, choose **Include source account metrics**.
4. Choose **Custom setup with Firehose**.
5. For **Select your Kinesis Data Firehose stream**, select the Firehose delivery stream to use. It must be in the same account. The default format for this option is OpenTelemetry 0.7.0, but you can change the format later in this procedure.

   Then select the Firehose delivery stream to use under **Select your Firehose delivery stream**.
6. (Optional)You can choose **Select existing service role** to use an existing IAM role instead of having CloudWatch create a new one for you.
7. (Optional) To change the output format from the default format for your scenario, choose **Change output format**. The supported formats are JSON, OpenTelemetry 1.0.0, and OpenTelemetry 0.7.0.
8. For **Metrics to be streamed**, choose either **All metrics** or **Select metrics**.

If you choose **All metrics**, all metrics from this account will be included in the stream.

Consider carefully whether to stream all metrics, because the more metrics that you stream the higher your metric stream charges will be.

If you choose **Select metrics**, do one of the following:

- To stream most metric namespaces, choose **Exclude** and select the namespaces or metrics to exclude. When you specify a namespace in **Exclude**, you can optionally select some specific metrics from that namespace to exclude. If you choose to exclude a namespace but don't then select metrics in that namespace, all metrics from that namespace are excluded.
- To include only a few metric namespaces or metrics in the metric stream, choose **Include** and then select the namespaces or metrics to include. If you choose to include a namespace but don't then select metrics in that namespace, all metrics from that namespace are included.

9. (Optional) To stream additional statistics for some of these metrics beyond Minimum, Maximum, SampleCount, and Sum, choose **Add additional statistics**. Either choose **Add recommended metrics** to add some commonly used statistics, or manually select the namespace and metric name to stream additional statistics for. Next, select the additional statistics to stream.

   To then choose another group of metrics to stream a different set of additional statistics for, choose **Add additional statistics**. Each metric can include as many as 20 additional statistics, and as many as 100 metrics within a metric stream can include additional statistics.

   Streaming additional statistics incurs more charges. For more information, see Statistics that can be streamed.

   For definitions of the additional statistics, see CloudWatch statistics definitions.
10. (Optional) Customize the name of the new metric stream under **Metric stream name**.
11. Choose **Create metric stream**.

Amazon CLI or Amazon API

Use the following steps to create a CloudWatch metric stream.

To use the Amazon CLI or Amazon API to create a metric stream

1. If you're streaming to Amazon S3, first create the bucket. For more information, see Creating a bucket.
2. Create the Firehose delivery stream. For more information, see Creating a Firehose stream.
3. Create an IAM role that enables CloudWatch to write to the Firehose delivery stream. For more information about the contents of this role, see Trust between CloudWatch and Firehose.
4. Use the `aws cloudwatch put-metric-stream` CLI command or the `PutMetricStream` API to create the CloudWatch metric stream.

Amazon CloudFormation

You can use Amazon CloudFormation to set up a metric stream. For more information, see AWS::CloudWatch::MetricStream.

To use Amazon CloudFormation to create a metric stream

1. If you're streaming to Amazon S3, first create the bucket. For more information, see Creating a bucket.
2. Create the Firehose delivery stream. For more information, see Creating a Firehose stream.
3. Create an IAM role that enables CloudWatch to write to the Firehose delivery stream. For more information about the contents of this role, see Trust between CloudWatch and Firehose.
4. Create the stream in Amazon CloudFormation. For more information, see AWS::CloudWatch::MetricStream.

Amazon Cloud Development Kit (Amazon CDK)

You can use Amazon Cloud Development Kit (Amazon CDK) to set up a metric stream.

To use the Amazon CDK to create a metric stream

1. If you're streaming to Amazon S3, first create the bucket. For more information, see Creating a bucket.
2. Create the Firehose delivery stream. For more information, see Creating an Amazon Data Firehose Delivery Stream.
3. Create an IAM role that enables CloudWatch to write to the Firehose delivery stream. For more information about the contents of this role, see Trust between CloudWatch and Firehose.

4. Create the metric stream. The metric stream resource is available in Amazon CDK as a Level 1 (L1) Construct named `CfnMetricStream`. For more information, see [Using L1 constructs](#).

Use Quick Amazon S3 setup

The **Quick S3 Setup** method works well if you want to quickly set up a stream to Amazon S3 and you don't need any formatting transformation beyond the supported JSON, OpenTelemetry 1.0.0, and OpenTelemetry 0.7.0 formats. CloudWatch will create all necessary resources including the Firehose delivery stream and the necessary IAM roles. The default format for this option is JSON, but you can change the format while you set up the stream.

Alternatively, if you want the final format to be Parquet format or Optimized Row Columnar (ORC), you should instead follow the steps in Custom setup with Firehose.

CloudWatch console

This section describes how to use the CloudWatch console to set up a metric stream Amazon S3 using Quick S3 setup.

To set up a metric stream using Quick S3 setup

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Metrics**, **Streams**. Then choose **Create metric stream**.
3. (Optional) If you are signed in to an account that is set up as a monitoring account in CloudWatch cross-account observability, you can choose whether to include metrics from linked source accounts in this metric stream. To include metrics from source accounts, choose **Include source account metrics**.
4. Choose **Quick S3 setup**. CloudWatch will create all necessary resources including the Firehose delivery stream and the necessary IAM roles. The default format for this option is JSON, but you can change the format later in this procedure.
5. (Optional) Choose **Select existing resources** to use an existing S3 bucket or existing IAM roles instead of having CloudWatch create new ones for you.
6. (Optional) To change the output format from the default format for your scenario, choose **Change output format**. The supported formats are JSON, OpenTelemetry 1.0.0, and OpenTelemetry 0.7.0.
7. For **Metrics to be streamed**, choose either **All metrics** or **Select metrics**.

   If you choose **All metrics**, all metrics from this account will be included in the stream.

Consider carefully whether to stream all metrics, because the more metrics that you stream the higher your metric stream charges will be.

If you choose **Select metrics**, do one of the following:

- To stream most metric namespaces, choose **Exclude** and select the namespaces or metrics to exclude. When you specify a namespace in **Exclude**, you can optionally select some specific metrics from that namespace to exclude. If you choose to exclude a namespace but don't then select metrics in that namespace, all metrics from that namespace are excluded.
- To include only a few metric namespaces or metrics in the metric stream, choose **Include** and then select the namespaces or metrics to include. If you choose to include a namespace but don't then select metrics in that namespace, all metrics from that namespace are included.

8. (Optional) To stream additional statistics for some of these metrics beyond Minimum, Maximum, SampleCount, and Sum, choose **Add additional statistics**. Either choose **Add recommended metrics** to add some commonly used statistics, or manually select the namespace and metric name to stream additional statistics for. Next, select the additional statistics to stream.

   To then choose another group of metrics to stream a different set of additional statistics for, choose **Add additional statistics**. Each metric can include as many as 20 additional statistics, and as many as 100 metrics within a metric stream can include additional statistics.

   Streaming additional statistics incurs more charges. For more information, see Statistics that can be streamed.

   For definitions of the additional statistics, see CloudWatch statistics definitions.
9. (Optional) Customize the name of the new metric stream under **Metric stream name**.
10. Choose **Create metric stream**.

Quick partner setup

CloudWatch provides a quick setup experience for the following third-party partners. To use this workflow, you need to provide only a destination URL and API key for your destination. CloudWatch handles the rest of setup including creating the Firehose delivery stream and the necessary IAM roles.

> **Important**
>
> Before you use quick partner setup to create a metric stream, we strongly recommend that you read that partner's documentation, linked in the following list.

- Datadog
- Dynatrace
- Elastic
- New Relic
- Splunk Observability Cloud
- SumoLogic

When you set up a metric stream to one of these partners, the stream is created with some default settings, as listed in the following sections.

Topics

- Set up a metric stream using quick partner setup
- Datadog stream defaults
- Dynatrace stream defaults
- Elastic stream defaults
- New Relic stream defaults
- Splunk Observability Cloud stream defaults
- Sumo Logic stream defaults

Set up a metric stream using quick partner setup

CloudWatch provides a quick setup option for some third-party partners. Before you start the steps in this section, you must have certain information for the partner. This information might include a destination URL and/or an API key for your partner destination. You should also read the documentation at the partner's website linked in the previous section, and the defaults for that partner listed in the following sections.

To stream to a third-party destination not supported by quick setup, you can follow the instructions in Follow the instructions in [Custom setup with Firehose](#) to set up a stream using Firehose, and then send those metrics from Firehose to the final destination.

To use quick partner setup to create a metric stream to third-party provider

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Metrics**, **Streams**. Then choose **Create metric stream**.
3. (Optional) If you are signed in to an account that is set up as a monitoring account in CloudWatch cross-account observability, you can choose whether to include metrics from linked source accounts in this metric stream. To include metrics from source accounts, choose **Include source account metrics**.
4. Choose **Quick Amazon Web Services partner setup**
5. Select the name of the partner that you want to stream metrics to.
6. For **Endpoint URL**, enter the destination URL.
7. For **Access Key** or **API Key**, enter the access key for the partner. Not all partners require an access key.
8. For **Metrics to be streamed**, choose either **All metrics** or **Select metrics**.

   If you choose **All metrics**, all metrics from this account will be included in the stream.

   Consider carefully whether to stream all metrics, because the more metrics that you stream the higher your metric stream charges will be.

   If you choose **Select metrics**, do one of the following:

   - To stream most metric namespaces, choose **Exclude** and select the namespaces or metrics to exclude. When you specify a namespace in **Exclude**, you can optionally select some specific metrics from that namespace to exclude. If you choose to exclude a namespace but don't then select metrics in that namespace, all metrics from that namespace are excluded.
   - To include only a few metric namespaces or metrics in the metric stream, choose **Include** and then select the namespaces or metrics to include. If you choose to include a namespace but don't then select metrics in that namespace, all metrics from that namespace are included.
9. (Optional) To stream additional statistics for some of these metrics beyond Minimum, Maximum, SampleCount, and Sum, choose **Add additional statistics**. Either choose **Add**

recommended metrics to add some commonly used statistics, or manually select the namespace and metric name to stream additional statistics for. Next, select the additional statistics to stream.

To then choose another group of metrics to stream a different set of additional statistics for, choose **Add additional statistics**. Each metric can include as many as 20 additional statistics, and as many as 100 metrics within a metric stream can include additional statistics.

Streaming additional statistics incurs more charges. For more information, see Statistics that can be streamed.

For definitions of the additional statistics, see CloudWatch statistics definitions.

10. (Optional) Customize the name of the new metric stream under **Metric stream name**.
11. Choose **Create metric stream**.

Datadog stream defaults

Quick partner setup streams to Datadog use the following defaults:

- **Output format:** OpenTelemetry 0.7.0
- **Firehose stream content encoding** GZIP
- **Firehose stream buffering options** Interval of 60 seconds, size of 4 MBs
- **Firehose stream retry option** Duration of 60 seconds

When you use quick partner setup to create a metric stream to Datadog and you stream certain metrics, by default those metrics include some additional statistics. Streaming additional statistics can incur additional charges. For more information about statistics and their charges, see Statistics that can be streamed.

The following list shows the metrics that have additional statistics streamed by default, if you choose to stream those metrics. You can choose to de-select these additional statistics before you start the stream.

- **`Duration` in `AWS/Lambda`:** p50, p80, p95, p99, p99.9
- **`PostRuntimeExtensionDuration` in `AWS/Lambda`:** p50, p99
- **`FirstByteLatency` and `TotalRequestLatency`in `AWS/S3`:** p50, p90, p95, p99, p99.9

- **`ResponseLatency` in `AWS/Polly` and `TargetResponseTime` in `AWS/ApplicationELB`:** p50, p90, p95, p99
- **`Latency` and `IntegrationLatency` in `AWS/ApiGateway`:** p90, p95, p99
- **`Latency` and `TargetResponseTime` in `AWS/ELB`:** p95, p99
- **`RequestLatency` in `AWS/AppRunner`:** p50, p95, p99
- **`ActivityTime`, `ExecutionTime`, `LambdaFunctionRunTime`, `LambdaFunctionScheduleTime`, `LambdaFunctionTime`, `ActivityRunTime`, and `ActivityScheduleTime` in `AWS/States`:** p95, p99
- **`EncoderBitRate`, `ConfiguredBitRate`, and `ConfiguredBitRateAvailable` in `AWS/MediaLive`:** p90
- **`Latency` in `AWS/AppSync`:** p90

Dynatrace stream defaults

Quick partner setup streams to Dynatrace use the following defaults:

- **Output format:** OpenTelemetry 0.7.0
- **Firehose stream content encoding** GZIP
- **Firehose stream buffering options** Interval of 60 seconds, size of 5 MBs
- **Firehose stream retry option** Duration of 600 seconds

Elastic stream defaults

Quick partner setup streams to Elastic use the following defaults:

- **Output format:** OpenTelemetry 1.0.0
- **Firehose stream content encoding** GZIP
- **Firehose stream buffering options** Interval of 60 seconds, size of 1 MB
- **Firehose stream retry option** Duration of 60 seconds

New Relic stream defaults

Quick partner setup streams to New Relic use the following defaults:

- **Output format:** OpenTelemetry 0.7.0

- **Firehose stream content encoding** GZIP
- **Firehose stream buffering options** Interval of 60 seconds, size of 1 MB
- **Firehose stream retry option** Duration of 60 seconds

Splunk Observability Cloud stream defaults

Quick partner setup streams to Splunk Observability Cloud use the following defaults:

- **Output format:** OpenTelemetry 1.0.0
- **Firehose stream content encoding** GZIP
- **Firehose stream buffering options** Interval of 60 seconds, size of 1 MB
- **Firehose stream retry option** Duration of 300 seconds

Sumo Logic stream defaults

Quick partner setup streams to Sumo Logic use the following defaults:

- **Output format:** OpenTelemetry 0.7.0
- **Firehose stream content encoding** GZIP
- **Firehose stream buffering options** Interval of 60 seconds, size of 1 MB
- **Firehose stream retry option** Duration of 60 seconds

Statistics that can be streamed

Metric streams always include the following statistics: `Minimum`, `Maximum`, `SampleCount`, and `Sum`. You can also choose to include the following additional statistics in a metric stream. This choice is on a per-metric basis. For more information about these statistics, see CloudWatch statistics definitions.

- Percentile values such as p95 or p99 (For streams with either JSON or OpenTelemetry format)
- Trimmed mean (Only for streams with the JSON format)
- Winsorized mean (Only for streams with the JSON format)
- Trimmed count (Only for streams with the JSON format)
- Trimmed sum (Only for streams with the JSON format)
- Percentile rank (Only for streams with the JSON format)

- Interquartile mean (Only for streams with the JSON format)

Streaming additional statistics incurs additional charges. Streaming between one and five of these additional statistics for a particular metric is billed as an additional metric update. Thereafter, each additional set of up to five of these statistics is billed as another metric update.

For example, suppose that for one metric you are streaming the following six additional statistics: p95, p99, p99.9, Trimmed mean, Winsorized mean, and Trimmed sum. Each update of this metric is billed as three metric updates: one for the metric update which includes the default statistics, one for the first five additional statistics, and one for the sixth additional statistic. Adding up to four more additional statistics for a total of ten would not increase the billing, but an eleventh additional statistic would do so.

When you specify a metric name and namespace combination to stream additional statistics, all dimension combinations of that metric name and namespace are streamed with the additional statistics.

CloudWatch metric streams publishes a new metric, `TotalMetricUpdate`, which reflects the base number of metric updates plus extra metric updates incurred by streaming additional statistics. For more information, see Monitoring your metric streams with CloudWatch metrics.

For more information, see Amazon CloudWatch Pricing.

> **Note**
>
> Some metrics do not support percentiles. Percentile statistics for these metrics are excluded from the stream and do not incur metric stream charges. An example of these statistics that do not support percentiles are some metrics in the `AWS/ECS` namespace.

The additional statistics that you configure are streamed only if they match the filters for the stream. For example, if you create a stream that has only `EC2` and `RDS` in the include filters, and then your statistics configuration lists `EC2` and `Lambda`, then the stream includes `EC2` metrics with additional statistics, `RDS` metrics with only the default statistics, and doesn't include `Lambda` statistics at all.

Metric stream operation and maintenance

Metric streams are always in one of two states, **Running** or **Stopped**.

- **Running**— The metric stream is running correctly. There might not be any metric data streamed to the destination because of the filters on the stream.
- **Stopped**— The metric stream has been explicitly stopped by someone, and not because of an error. It might be useful to stop your stream to temporarily pause the streaming of data without deleting the stream.

If you stop and restart a metric stream, the metric data that was published to CloudWatch while the metric stream was stopped is not backfilled to the metric stream.

If you change the output format of a metric stream, in certain cases you might see a small amount of metric data written to the destination in both the old format and the new format. To avoid this situation, you can create a new Firehose delivery stream with the same configuration as your current one, then change to the new Firehose delivery stream and change the output format at the same time. This way, the Kinesis records with different output format are stored on Amazon S3 in separate objects. Later, you can direct the traffic back to the original Firehose delivery stream and delete the second delivery stream.

To view, edit, stop, and start your metric streams

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Metrics**, **Streams**.

   The list of streams appears, and the **Status** column displays whether each stream is running or stopped.
3. To stop or start a metric stream, select the stream and choose **Stop** or **Start**.
4. To see the details about a metric stream, select the stream and choose **View details**.
5. To change the stream's output format, filters, destination Firehose stream, or roles, choose **Edit** and make the changes that you want.

   If you change the filters, there might be some gaps in the metric data during the transition.

Monitoring your metric streams with CloudWatch metrics

Metric streams emit CloudWatch metrics about their health and operation in the `AWS/CloudWatch/MetricStreams` namespace. The following metrics are emitted. These metrics are emitted with a `MetricStreamName` dimension and with no dimension. You can use the metrics with no dimensions to see aggregated metrics for all of your metric streams. You can use the metrics with the `MetricStreamName` dimension to see the metrics about only that metric stream.

For all of these metrics, values are emitted only for metric streams that are in the **Running** state.

Metric	Description
`MetricUpdate`	The number of metric updates sent to the metric stream. If no metric updates are streamed during a time period, this metric is not emitted during that time period. If you stop the metric stream, this metric stops being emitted until the metric stream is started again. Valid Statistic: `Sum` Units: None
`TotalMetricUpdate`	This is calculated as **MetricUpdate + a number based on additional statistics that are being streamed**. For each unique namespace and metric name combination, streaming 1-5 additional statistics adds 1 to the `TotalMetricUpdate` , streaming 6-10 additional statistics adds 2 to `TotalMetricUpdate` , and so on. Valid Statistic: `Sum` Units: None
`PublishErrorRate`	The number of unrecoverable errors that occur when putting data into the Firehose delivery stream. If no errors occur during a time period, this metric is not emitted during that time period. If you stop the metric stream, this metric stops being emitted until the metric stream is started again. Valid Statistic: `Average` to see the rate of metric updates unable to be written. This value will be between 0.0 and 1.0.

Metric	Description
	Units: None

Trust between CloudWatch and Firehose

The Firehose delivery stream must trust CloudWatch through an IAM role that has write permissions to Firehose. These permissions can be limited to the single Firehose delivery stream that the CloudWatch metric stream uses. The IAM role must trust the `streams.metrics.cloudwatch.amazonaws.com` service principal.

If you use the CloudWatch console to create a metric stream, you can have CloudWatch create the role with the correct permissions. If you use another method to create a metric stream, or you want to create the IAM role itself, it must contain the following permissions policy and trust policy.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "firehose:PutRecord",
                "firehose:PutRecordBatch"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:firehose:region:account-id:deliverystream/*"
        }
    ]
}
```

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "streams.metrics.cloudwatch.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
```

```
}
```

Metric data is streamed by CloudWatch to the destination Firehose delivery stream on behalf of the source that owns the metric stream resource.

CloudWatch metric stream output in JSON format

In a CloudWatch metric stream that uses the JSON format, each Firehose record contains multiple JSON objects separated by a newline character (\n). Each object includes a single data point of a single metric.

The JSON format that is used is fully compatible with Amazon Glue and with Amazon Athena. If you have a Firehose delivery stream and an Amazon Glue table formatted correctly, the format can be automatically transformed into Parquet format or Optimized Row Columnar (ORC) format before being stored in S3. For more information about transforming the format, see Converting Your Input Record Format in Firehose. For more information about the correct format for Amazon Glue, see Which Amazon Glue schema should I use for JSON output format?.

In the JSON format, the valid values for `unit` are the same as for the value of `unit` in the `MetricDatum` API structure. For more information, see MetricDatum. The value for the `timestamp` field is in epoch milliseconds, such as `1616004674229`.

The following is an example of the format. In this example, the JSON is formatted for easy reading, but in practice the whole format is on a single line.

```
{
    "metric_stream_name": "MyMetricStream",
    "account_id": "1234567890",
    "region": "us-east-1",
    "namespace": "AWS/EC2",
    "metric_name": "DiskWriteOps",
    "dimensions": {
        "InstanceId": "i-123456789012"
    },
    "timestamp": 1611929698000,
    "value": {
        "count": 3.0,
        "sum": 20.0,
        "max": 18.0,
        "min": 0.0,
```

```
        "p99": 17.56,
        "p99.9": 17.8764,
        "TM(25%:75%)": 16.43
    },
    "unit": "Seconds"
}
```

Which Amazon Glue schema should I use for JSON output format?

The following is an example of a JSON representation of the `StorageDescriptor` for an Amazon Glue table, which would then be used by Firehose. For more information about `StorageDescriptor`, see StorageDescriptor.

```
{
  "Columns": [
    {
      "Name": "metric_stream_name",
      "Type": "string"
    },
    {
      "Name": "account_id",
      "Type": "string"
    },
    {
      "Name": "region",
      "Type": "string"
    },
    {
      "Name": "namespace",
      "Type": "string"
    },
    {
      "Name": "metric_name",
      "Type": "string"
    },
    {
      "Name": "timestamp",
      "Type": "timestamp"
    },
    {
      "Name": "dimensions",
      "Type": "map<string,string>"
    },
```

```
    {
      "Name": "value",
      "Type":
 "struct<min:double,max:double,count:double,sum:double,p99:double,p99.9:double>"
    },
    {
      "Name": "unit",
      "Type": "string"
    }
  ],
  "Location": "s3://amzn-s3-demo-bucket/",
  "InputFormat": "org.apache.hadoop.mapred.TextInputFormat",
  "OutputFormat": "org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat",
  "SerdeInfo": {
    "SerializationLibrary": "org.apache.hive.hcatalog.data.JsonSerDe"
  },
  "Parameters": {
    "classification": "json"
  }
}
```

The preceding example is for data written on Amazon S3 in JSON format. Replace the values in the following fields with the indicated values to store the data in Parquet format or Optimized Row Columnar (ORC) format.

- **Parquet:**
  - inputFormat: org.apache.hadoop.hive.ql.io.parquet.MapredParquetInputFormat
  - outputFormat: org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat
  - SerDeInfo.serializationLib: org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe
  - parameters.classification: parquet
- **ORC:**
  - inputFormat: org.apache.hadoop.hive.ql.io.orc.OrcInputFormat
  - outputFormat: org.apache.hadoop.hive.ql.io.orc.OrcOutputFormat
  - SerDeInfo.serializationLib: org.apache.hadoop.hive.ql.io.orc.OrcSerde
  - parameters.classification: orc

CloudWatch metric stream output in OpenTelemetry 1.0.0 format

Note

With the OpenTelemetry 1.0.0 format, metric attributes are encoded as a list of `KeyValue` objects instead of the `StringKeyValue` type used in the 0.7.0 format. As a consumer, this is the only major change between the 0.7.0 and 1.0.0 formats. A parser generated from the 0.7.0 proto files won't parse metric attributes encoded in the 1.0.0 format. The same is true in reverse, a parser generated from the 1.0.0 proto files will not parse metric attributes encoded in the 0.7.0 format.

OpenTelemetry is a collection of tools, APIs, and SDKs. You can use it to instrument, generate, collect, and export telemetry data (metrics, logs, and traces) for analysis. OpenTelemetry is part of the Cloud Native Computing Foundation. For more information, see OpenTelemetry.

For information about the full OpenTelemetry 1.0.0 specification, see Release version 1.0.0.

A Kinesis record can contain one or more `ExportMetricsServiceRequest` OpenTelemetry data structures. Each data structure starts with a header with an `UnsignedVarInt32` indicating the record length in bytes. Each `ExportMetricsServiceRequest` may contain data from multiple metrics at once.

The following is a string representation of the message of the `ExportMetricsServiceRequest` OpenTelemetry data structure. OpenTelemetry serializes the Google Protocol Buffers binary protocol, and this is not human-readable.

```
resource_metrics {
  resource {
    attributes {
      key: "cloud.provider"
      value {
        string_value: "aws"
      }
    }
    attributes {
      key: "cloud.account.id"
      value {
        string_value: "123456789012"
      }
```

```
    }
    attributes {
      key: "cloud.region"
      value {
        string_value: "us-east-1"
      }
    }
    attributes {
      key: "aws.exporter.arn"
      value {
        string_value: "arn:aws:cloudwatch:us-east-1:123456789012:metric-stream/
MyMetricStream"
      }
    }
  }
  scope_metrics {
    metrics {
      name: "amazonaws.com/AWS/DynamoDB/ConsumedReadCapacityUnits"
      unit: "NoneTranslated"
      summary {
        data_points {
          start_time_unix_nano: 60000000000
          time_unix_nano: 120000000000
          count: 1
          sum: 1.0
          quantile_values {
            value: 1.0
          }
          quantile_values {
            quantile: 0.95
            value: 1.0
          }
          quantile_values {
            quantile: 0.99
            value: 1.0
          }
          quantile_values {
            quantile: 1.0
            value: 1.0
          }
          attributes {
            key: "Namespace"
            value {
              string_value: "AWS/DynamoDB"
```

```
          }
        }
        attributes {
          key: "MetricName"
          value {
            string_value: "ConsumedReadCapacityUnits"
          }
        }
        attributes {
          key: "Dimensions"
          value {
            kvlist_value {
              values {
                key: "TableName"
                value {
                  string_value: "MyTable"
                }
              }
            }
          }
        }
      }
      data_points {
        start_time_unix_nano: 70000000000
        time_unix_nano: 130000000000
        count: 2
        sum: 5.0
        quantile_values {
          value: 2.0
        }
        quantile_values {
          quantile: 1.0
          value: 3.0
        }
        attributes {
          key: "Namespace"
          value {
            string_value: "AWS/DynamoDB"
          }
        }
        attributes {
          key: "MetricName"
          value {
            string_value: "ConsumedReadCapacityUnits"
```

```
          }
        }
        attributes {
          key: "Dimensions"
          value {
            kvlist_value {
              values {
                key: "TableName"
                value {
                  string_value: "MyTable"
                }
              }
            }
          }
        }
      }
    }
  }
 }
}
```

Top-level object to serialize OpenTelemetry metric data

`ExportMetricsServiceRequest` is the top-level wrapper to serialize an OpenTelemetry exporter payload. It contains one or more `ResourceMetrics`.

```
message ExportMetricsServiceRequest {
  // An array of ResourceMetrics.
  // For data coming from a single resource this array will typically contain one
  // element. Intermediary nodes (such as OpenTelemetry Collector) that receive
  // data from multiple origins typically batch the data before forwarding further and
  // in that case this array will contain multiple elements.
  repeated opentelemetry.proto.metrics.v1.ResourceMetrics resource_metrics = 1;
}
```

`ResourceMetrics` is the top-level object to represent MetricData objects.

```
// A collection of ScopeMetrics from a Resource.
message ResourceMetrics {
  reserved 1000;

  // The resource for the metrics in this message.
```

```
  // If this field is not set then no resource info is known.
  opentelemetry.proto.resource.v1.Resource resource = 1;

  // A list of metrics that originate from a resource.
  repeated ScopeMetrics scope_metrics = 2;

  // This schema_url applies to the data in the "resource" field. It does not apply
  // to the data in the "scope_metrics" field which have their own schema_url field.
  string schema_url = 3;
}
```

The Resource object

A `Resource` object is a value-pair object that contains some information about the resource that generated the metrics. For metrics created by Amazon, the data structure contains the Amazon Resource Name (ARN) of the resource related to the metric, such as an EC2 instance or an S3 bucket.

The `Resource` object contains an attribute called `attributes`, which store a list of key-value pairs.

- `cloud.account.id` contains the account ID
- `cloud.region` contains the Region
- `aws.exporter.arn` contains the metric stream ARN
- `cloud.provider` is always `aws`.

```
// Resource information.
message Resource {
  // Set of attributes that describe the resource.
  // Attribute keys MUST be unique (it is not allowed to have more than one
  // attribute with the same key).
  repeated opentelemetry.proto.common.v1.KeyValue attributes = 1;

  // dropped_attributes_count is the number of dropped attributes. If the value is 0,
 then
  // no attributes were dropped.
  uint32 dropped_attributes_count = 2;
}
```

The ScopeMetrics object

The scope field will not be filled. We fill only the metrics field that we are exporting.

```
// A collection of Metrics produced by an Scope.
message ScopeMetrics {
  // The instrumentation scope information for the metrics in this message.
  // Semantically when InstrumentationScope isn't set, it is equivalent with
  // an empty instrumentation scope name (unknown).
  opentelemetry.proto.common.v1.InstrumentationScope scope = 1;

  // A list of metrics that originate from an instrumentation library.
  repeated Metric metrics = 2;

  // This schema_url applies to all metrics in the "metrics" field.
  string schema_url = 3;
}
```

The Metric object

The metric object contains some metadata and a `Summary` data field that contains a list of `SummaryDataPoint`.

For metric streams, the metadata is as follows:

- name will be `amazonaws.com/`*`metric_namespace`*`/`*`metric_name`*
- `description` will be blank
- `unit` will be filled by mapping the metric datum's unit to the case-sensitive variant of the Unified code for Units of Measure. For more information, see [Translations with OpenTelemetry 1.0.0 format in CloudWatch] and [The Unified Code For Units of Measure].
- `type` will be SUMMARY

```
message Metric {
  reserved 4, 6, 8;

  // name of the metric, including its DNS name prefix. It must be unique.
  string name = 1;

  // description of the metric, which can be used in documentation.
  string description = 2;

  // unit in which the metric value is reported. Follows the format
```

```
  // described by http://unitsofmeasure.org/ucum.html.
  string unit = 3;

  // Data determines the aggregation type (if any) of the metric, what is the
  // reported value type for the data points, as well as the relatationship to
  // the time interval over which they are reported.
  oneof data {
    Gauge gauge = 5;
    Sum sum = 7;
    Histogram histogram = 9;
    ExponentialHistogram exponential_histogram = 10;
    Summary summary = 11;
  }
}

message Summary {
  repeated SummaryDataPoint data_points = 1;
}
```

The SummaryDataPoint object

The SummaryDataPoint object contains the value of a single data point in a time series in a DoubleSummary metric.

```
// SummaryDataPoint is a single data point in a timeseries that describes the
// time-varying values of a Summary metric.
message SummaryDataPoint {
  reserved 1;

  // The set of key/value pairs that uniquely identify the timeseries from
  // where this point belongs. The list may be empty (may contain 0 elements).
  // Attribute keys MUST be unique (it is not allowed to have more than one
  // attribute with the same key).
  repeated opentelemetry.proto.common.v1.KeyValue attributes = 7;

  // StartTimeUnixNano is optional but strongly encouraged, see the
  // the detailed comments above Metric.
  //
  // Value is UNIX Epoch time in nanoseconds since 00:00:00 UTC on 1 January
  // 1970.
  fixed64 start_time_unix_nano = 2;

  // TimeUnixNano is required, see the detailed comments above Metric.
```

```
  //
  // Value is UNIX Epoch time in nanoseconds since 00:00:00 UTC on 1 January
  // 1970.
  fixed64 time_unix_nano = 3;

  // count is the number of values in the population. Must be non-negative.
  fixed64 count = 4;

  // sum of the values in the population. If count is zero then this field
  // must be zero.
  //
  // Note: Sum should only be filled out when measuring non-negative discrete
  // events, and is assumed to be monotonic over the values of these events.
  // Negative events *can* be recorded, but sum should not be filled out when
  // doing so.  This is specifically to enforce compatibility w/ OpenMetrics,
  // see: https://github.com/OpenObservability/OpenMetrics/blob/main/specification/
OpenMetrics.md#summary
  double sum = 5;

  // Represents the value at a given quantile of a distribution.
  //
  // To record Min and Max values following conventions are used:
  // - The 1.0 quantile is equivalent to the maximum value observed.
  // - The 0.0 quantile is equivalent to the minimum value observed.
  //
  // See the following issue for more context:
  // https://github.com/open-telemetry/opentelemetry-proto/issues/125
  message ValueAtQuantile {
    // The quantile of a distribution. Must be in the interval
    // [0.0, 1.0].
    double quantile = 1;

    // The value at the given quantile of a distribution.
    //
    // Quantile values must NOT be negative.
    double value = 2;
  }

  // (Optional) list of values at different quantiles of the distribution calculated
  // from the current snapshot. The quantiles must be strictly increasing.
  repeated ValueAtQuantile quantile_values = 6;

  // Flags that apply to this specific data point.  See DataPointFlags
  // for the available flags and their meaning.
```

```
  uint32 flags = 8;
}
```

For more information, see [Translations with OpenTelemetry 1.0.0 format in CloudWatch](#).

Translations with OpenTelemetry 1.0.0 format in CloudWatch

CloudWatch performs some transformations to put CloudWatch data into OpenTelemetry format.

Translating namespace, metric name, and dimensions

These attributes are key-value pairs encoded in the mapping.

- One attribute has the key `Namespace` and its value is the namespace of the metric
- One attribute has the key `MetricName` and its value is the name of the metric
- One pair has the key `Dimensions` and its value is a nested list of key-value pairs. Each pair in this list maps to a CloudWatch metric dimension, where the pair's key is the name of the dimension and its value is the value of the dimension.

Translating Average, Sum, SampleCount, Min and Max

The Summary datapoint enables CloudWatch to export all of these statistics using one datapoint.

- `startTimeUnixNano` contains the CloudWatch `startTime`
- `timeUnixNano` contains the CloudWatch `endTime`
- `sum` contains the Sum statistic.
- `count` contains the SampleCount statistic.
- `quantile_values` contains two `valueAtQuantile.value` objects:
  - `valueAtQuantile.quantile = 0.0` with `valueAtQuantile.value = Min value`
  - `valueAtQuantile.quantile = 0.99` with `valueAtQuantile.value = p99 value`
  - `valueAtQuantile.quantile = 0.999` with `valueAtQuantile.value = p99.9 value`
  - `valueAtQuantile.quantile = 1.0` with `valueAtQuantile.value = Max value`

Resources that consume the metric stream can calculate the Average statistic as **Sum/ SampleCount**.

Translating units

CloudWatch units are mapped to the case-sensitive variant of the Unified code for Units of Measure, as shown in the following table. For more information, see The Unified Code For Units of Measure.

CloudWatch	OpenTelemetry
Second	s
Second or Seconds	s
Microseconds	us
Milliseconds	ms
Bytes	By
Kilobytes	kBy
Megabytes	MBy
Gigabytes	GBy
Terabytes	TBy
Bits	bit
Kilobits	kbit
Megabits	MBit
Gigabits	GBit
Terabits	Tbit
Percent	%
Count	{Count}
None	1

Units that are combined with a slash are mapped by applying the OpenTelemetry conversion of both the units. For example, Bytes/Second is mapped to By/s.

How to parse OpenTelemetry 1.0.0 messages

This section provides information to help you get started with parsing OpenTelemetry 1.0.0.

First, you should get language-specific bindings, which enable you to parse OpenTelemetry 1.0.0 messages in your preferred language.

To get language-specific bindings

- The steps depend on your preferred language.
  - To use Java, add the following Maven dependency to your Java project: OpenTelemetry Java >> 0.14.1.
  - To use any other language, follow these steps:
    a. Make sure that your language is supported by checking the list at Generating Your Classes.
    b. Install the Protobuf compiler by following the steps at Download Protocol Buffers.
    c. Download the OpenTelemetry 1.0.0 ProtoBuf definitions at Release version 1.0.0.
    d. Confirm that you are in the root folder of the downloaded OpenTelemetry 1.0.0 ProtoBuf definitions. Then create a `src` folder and then run the command to generate language-specific bindings. For more information, see Generating Your Classes.

       The following is an example for how to generate Javascript bindings.

```
protoc --proto_path=./ --js_out=import_style=commonjs,binary:src \
opentelemetry/proto/common/v1/common.proto \
opentelemetry/proto/resource/v1/resource.proto \
opentelemetry/proto/metrics/v1/metrics.proto \
opentelemetry/proto/collector/metrics/v1/metrics_service.proto
```

The following section includes examples of using the language-specific bindings that you can build using the previous instructions.

Java

```
package com.example;

import io.opentelemetry.proto.collector.metrics.v1.ExportMetricsServiceRequest;

import java.io.IOException;
import java.io.InputStream;
import java.util.ArrayList;
import java.util.List;

public class MyOpenTelemetryParser {

    public List<ExportMetricsServiceRequest> parse(InputStream inputStream) throws
 IOException {
        List<ExportMetricsServiceRequest> result = new ArrayList<>();

        ExportMetricsServiceRequest request;
        /* A Kinesis record can contain multiple `ExportMetricsServiceRequest`
           records, each of them starting with a header with an
           UnsignedVarInt32 indicating the record length in bytes:
            ------ --------------------------- ------ -----------------------
           |UINT32|ExportMetricsServiceRequest|UINT32|ExportMetricsService...
            ------ --------------------------- ------ -----------------------
         */
        while ((request =
 ExportMetricsServiceRequest.parseDelimitedFrom(inputStream)) != null) {
            // Do whatever we want with the parsed message
            result.add(request);
        }

        return result;
    }
}
```

Javascript

This example assumes that the root folder with the bindings generated is `./`

The data argument of the function `parseRecord` can be one of the following types:

- `Uint8Array` this is optimal
- `Buffer` optimal under node
- `Array.`*`number`* 8-bit integers

```
const pb = require('google-protobuf')
const pbMetrics =
    require('./opentelemetry/proto/collector/metrics/v1/metrics_service_pb')

function parseRecord(data) {
    const result = []

    // Loop until we've read all the data from the buffer
    while (data.length) {
        /* A Kinesis record can contain multiple `ExportMetricsServiceRequest`
           records, each of them starting with a header with an
           UnsignedVarInt32 indicating the record length in bytes:
            ------ -------------------------- ------ -----------------------
           |UINT32|ExportMetricsServiceRequest|UINT32|ExportMetricsService...
            ------ -------------------------- ------ -----------------------
         */
        const reader = new pb.BinaryReader(data)
        const messageLength = reader.decoder_.readUnsignedVarint32()
        const messageFrom = reader.decoder_.cursor_
        const messageTo = messageFrom + messageLength

        // Extract the current `ExportMetricsServiceRequest` message to parse
        const message = data.subarray(messageFrom, messageTo)

        // Parse the current message using the ProtoBuf library
        const parsed =
            pbMetrics.ExportMetricsServiceRequest.deserializeBinary(message)

        // Do whatever we want with the parsed message
        result.push(parsed.toObject())

        // Shrink the remaining buffer, removing the already parsed data
        data = data.subarray(messageTo)
    }

    return result
}
```

Python

You must read the `var-int` delimiters yourself or use the internal methods `_VarintBytes(size)` and `_DecodeVarint32(buffer, position)`. These return the position in the buffer just after the size bytes. The read-side constructs a new buffer that is limited to reading only the bytes of the message.

```
size = my_metric.ByteSize()
f.write(_VarintBytes(size))
f.write(my_metric.SerializeToString())
msg_len, new_pos = _DecodeVarint32(buf, 0)
msg_buf = buf[new_pos:new_pos+msg_len]
request = metrics_service_pb.ExportMetricsServiceRequest()
request.ParseFromString(msg_buf)
```

Go

Use `Buffer.DecodeMessage()`.

C#

Use `CodedInputStream`. This class can read size-delimited messages.

C++

The functions described in `google/protobuf/util/delimited_message_util.h` can read size-delimited messages.

Other languages

For other languages, see [Download Protocol Buffers].

When implementing the parser, consider that a Kinesis record can contain multiple `ExportMetricsServiceRequest` Protocol Buffers messages, each of them starting with a header with an `UnsignedVarInt32` that indicates the record length in bytes.

CloudWatch metric stream output in OpenTelemetry 0.7.0 format

OpenTelemetry is a collection of tools, APIs, and SDKs. You can use it to instrument, generate, collect, and export telemetry data (metrics, logs, and traces) for analysis. OpenTelemetry is part of the Cloud Native Computing Foundation. For more information, see [OpenTelemetry].

For information about the full OpenTelemetry 0.7.0 specification, see [v0.7.0 release].

A Kinesis record can contain one or more `ExportMetricsServiceRequest` OpenTelemetry data structures. Each data structure starts with a header with an `UnsignedVarInt32` indicating the record length in bytes. Each `ExportMetricsServiceRequest` may contain data from multiple metrics at once.

The following is a string representation of the message of the `ExportMetricsServiceRequest` OpenTelemetry data structure. OpenTelemetry serializes the Google Protocol Buffers binary protocol, and this is not human-readable.

```
resource_metrics {
  resource {
    attributes {
      key: "cloud.provider"
      value {
        string_value: "aws"
      }
    }
    attributes {
      key: "cloud.account.id"
      value {
        string_value: "2345678901"
      }
    }
    attributes {
      key: "cloud.region"
      value {
        string_value: "us-east-1"
      }
    }
    attributes {
      key: "aws.exporter.arn"
      value {
        string_value: "arn:aws:cloudwatch:us-east-1:123456789012:metric-stream/
MyMetricStream"
      }
    }
  }
  instrumentation_library_metrics {
    metrics {
      name: "amazonaws.com/AWS/DynamoDB/ConsumedReadCapacityUnits"
      unit: "1"
      double_summary {
        data_points {
          labels {
            key: "Namespace"
```

```
        value: "AWS/DynamoDB"
      }
      labels {
        key: "MetricName"
        value: "ConsumedReadCapacityUnits"
      }
      labels {
        key: "TableName"
        value: "MyTable"
      }
      start_time_unix_nano: 1604948400000000000
      time_unix_nano: 1604948460000000000
      count: 1
      sum: 1.0
      quantile_values {
        quantile: 0.0
        value: 1.0
      }
      quantile_values {
        quantile: 0.95
        value: 1.0
      }
      quantile_values {
        quantile: 0.99
        value: 1.0
      }
      quantile_values {
        quantile: 1.0
        value: 1.0
      }
    }
    data_points {
      labels {
        key: "Namespace"
        value: "AWS/DynamoDB"
      }
      labels {
        key: "MetricName"
        value: "ConsumedReadCapacityUnits"
      }
      labels {
        key: "TableName"
        value: "MyTable"
      }
```

```
          start_time_unix_nano: 1604948460000000000
          time_unix_nano: 1604948520000000000
          count: 2
          sum: 5.0
          quantile_values {
            quantile: 0.0
            value: 2.0
          }
          quantile_values {
            quantile: 1.0
            value: 3.0
          }
        }
      }
    }
  }
}
```

Top-level object to serialize OpenTelemetry metric data

`ExportMetricsServiceRequest` is the top-level wrapper to serialize an OpenTelemetry exporter payload. It contains one or more `ResourceMetrics`.

```
message ExportMetricsServiceRequest {
  // An array of ResourceMetrics.
  // For data coming from a single resource this array will typically contain one
  // element. Intermediary nodes (such as OpenTelemetry Collector) that receive
  // data from multiple origins typically batch the data before forwarding further and
  // in that case this array will contain multiple elements.
  repeated opentelemetry.proto.metrics.v1.ResourceMetrics resource_metrics = 1;
}
```

`ResourceMetrics` is the top-level object to represent MetricData objects.

```
// A collection of InstrumentationLibraryMetrics from a Resource.
message ResourceMetrics {
  // The resource for the metrics in this message.
  // If this field is not set then no resource info is known.
  opentelemetry.proto.resource.v1.Resource resource = 1;

  // A list of metrics that originate from a resource.
  repeated InstrumentationLibraryMetrics instrumentation_library_metrics = 2;
```

```
}
```

The Resource object

A `Resource` object is a value-pair object that contains some information about the resource that generated the metrics. For metrics created by Amazon, the data structure contains the Amazon Resource Name (ARN) of the resource related to the metric, such as an EC2 instance or an S3 bucket.

The `Resource` object contains an attribute called `attributes`, which store a list of key-value pairs.

- `cloud.account.id` contains the account ID
- `cloud.region` contains the Region
- `aws.exporter.arn` contains the metric stream ARN
- `cloud.provider` is always `aws`.

```
// Resource information.
message Resource {
  // Set of labels that describe the resource.
  repeated opentelemetry.proto.common.v1.KeyValue attributes = 1;

  // dropped_attributes_count is the number of dropped attributes. If the value is 0,
  // no attributes were dropped.
  uint32 dropped_attributes_count = 2;
}
```

The InstrumentationLibraryMetrics object

The instrumentation_library field will not be filled. We will fill only the metrics field that we are exporting.

```
// A collection of Metrics produced by an InstrumentationLibrary.
message InstrumentationLibraryMetrics {
  // The instrumentation library information for the metrics in this message.
  // If this field is not set then no library info is known.
  opentelemetry.proto.common.v1.InstrumentationLibrary instrumentation_library = 1;
  // A list of metrics that originate from an instrumentation library.
  repeated Metric metrics = 2;
```

```
}
```

The Metric object

The metric object contains a `DoubleSummary` data field that contains a list of `DoubleSummaryDataPoint`.

```
message Metric {
  // name of the metric, including its DNS name prefix. It must be unique.
  string name = 1;

  // description of the metric, which can be used in documentation.
  string description = 2;

  // unit in which the metric value is reported. Follows the format
  // described by http://unitsofmeasure.org/ucum.html.
  string unit = 3;

  oneof data {
    IntGauge int_gauge = 4;
    DoubleGauge double_gauge = 5;
    IntSum int_sum = 6;
    DoubleSum double_sum = 7;
    IntHistogram int_histogram = 8;
    DoubleHistogram double_histogram = 9;
    DoubleSummary double_summary = 11;
  }
}

message DoubleSummary {
  repeated DoubleSummaryDataPoint data_points = 1;
}
```

The MetricDescriptor object

The MetricDescriptor object contains metadata. For more information, see metrics.proto on GitHub.

For metric streams, the MetricDescriptor has the following contents:

- name will be `amazonaws.com/`*`metric_namespace`*`/`*`metric_name`*
- `description` will be blank.

- `unit` will be filled by mapping the metric datum's unit to the case-sensitive variant of the Unified code for Units of Measure. For more information, see [Translations with OpenTelemetry 0.7.0 format in CloudWatch] and [The Unified Code For Units of Measure].
- `type` will be `SUMMARY`.

The DoubleSummaryDataPoint object

The DoubleSummaryDataPoint object contains the value of a single data point in a time series in a DoubleSummary metric.

```
// DoubleSummaryDataPoint is a single data point in a timeseries that describes the
// time-varying values of a Summary metric.
message DoubleSummaryDataPoint {
  // The set of labels that uniquely identify this timeseries.
  repeated opentelemetry.proto.common.v1.StringKeyValue labels = 1;

  // start_time_unix_nano is the last time when the aggregation value was reset
  // to "zero". For some metric types this is ignored, see data types for more
  // details.
  //
  // The aggregation value is over the time interval (start_time_unix_nano,
  // time_unix_nano].
  //
  // Value is UNIX Epoch time in nanoseconds since 00:00:00 UTC on 1 January
  // 1970.
  //
  // Value of 0 indicates that the timestamp is unspecified. In that case the
  // timestamp may be decided by the backend.
  fixed64 start_time_unix_nano = 2;

  // time_unix_nano is the moment when this aggregation value was reported.
  //
  // Value is UNIX Epoch time in nanoseconds since 00:00:00 UTC on 1 January
  // 1970.
  fixed64 time_unix_nano = 3;

  // count is the number of values in the population. Must be non-negative.
  fixed64 count = 4;

  // sum of the values in the population. If count is zero then this field
  // must be zero.
  double sum = 5;
```

```
  // Represents the value at a given quantile of a distribution.
  //
  // To record Min and Max values following conventions are used:
  // - The 1.0 quantile is equivalent to the maximum value observed.
  // - The 0.0 quantile is equivalent to the minimum value observed.
  message ValueAtQuantile {
    // The quantile of a distribution. Must be in the interval
    // [0.0, 1.0].
    double quantile = 1;

    // The value at the given quantile of a distribution.
    double value = 2;
  }

  // (Optional) list of values at different quantiles of the distribution calculated
  // from the current snapshot. The quantiles must be strictly increasing.
  repeated ValueAtQuantile quantile_values = 6;
}
```

For more information, see Translations with OpenTelemetry 0.7.0 format in CloudWatch.

Translations with OpenTelemetry 0.7.0 format in CloudWatch

CloudWatch performs some transformations to put CloudWatch data into OpenTelemetry format.

Translating namespace, metric name, and dimensions

These attributes are key-value pairs encoded in the mapping.

- One pair contains the namespace of the metric
- One pair contains the name of the metric
- For each dimension, CloudWatch stores the following pair: `metricDatum.Dimensions[i].`*`Name`*`, metricDatum.Dimensions[i].`*`Value`*

Translating Average, Sum, SampleCount, Min and Max

The Summary datapoint enables CloudWatch to export all of these statistics using one datapoint.

- `startTimeUnixNano` contains the CloudWatch `startTime`
- `timeUnixNano` contains the CloudWatch `endTime`

- `sum` contains the Sum statistic.
- `count` contains the SampleCount statistic.
- `quantile_values` contains two `valueAtQuantile.value` objects:
  - `valueAtQuantile.quantile = 0.0` with `valueAtQuantile.value = Min value`
  - `valueAtQuantile.quantile = 0.99` with `valueAtQuantile.value = p99 value`
  - `valueAtQuantile.quantile = 0.999` with `valueAtQuantile.value = p99.9 value`
  - `valueAtQuantile.quantile = 1.0` with `valueAtQuantile.value = Max value`

Resources that consume the metric stream can calculate the Average statistic as **Sum/ SampleCount**.

Translating units

CloudWatch units are mapped to the case-sensitive variant of the Unified code for Units of Measure, as shown in the following table. For more information, see [The Unified Code For Units of Measure](https://ucum.org/ucum).

CloudWatch	OpenTelemetry
Second	s
Second or Seconds	s
Microsecond	us
Milliseconds	ms
Bytes	By
Kilobytes	kBy
Megabytes	MBy
Gigabytes	GBy
Terabytes	TBy
Bits	bit

CloudWatch	OpenTelemetry
Kilobits	kbit
Megabits	MBit
Gigabits	GBit
Terabits	Tbit
Percent	%
Count	{Count}
None	1

Units that are combined with a slash are mapped by applying the OpenTelemetry conversion of both the units. For example, Bytes/Second is mapped to By/s.

How to parse OpenTelemetry 0.7.0 messages

This section provides information to help you get started with parsing OpenTelemetry 0.7.0.

First, you should get language-specific bindings, which enable you to parse OpenTelemetry 0.7.0 messages in your preferred language.

To get language-specific bindings

- The steps depend on your preferred language.
  - To use Java, add the following Maven dependency to your Java project: OpenTelemetry Java >> 0.14.1.
  - To use any other language, follow these steps:

    a. Make sure that your language is supported by checking the list at Generating Your Classes.

    b. Install the Protobuf compiler by following the steps at Download Protocol Buffers.

    c. Download the OpenTelemetry 0.7.0 ProtoBuf definitions at v0.7.0 release.

    d. Confirm that you are in the root folder of the downloaded OpenTelemetry 0.7.0 ProtoBuf definitions. Then create a `src` folder and then run the command to generate language-specific bindings. For more information, see Generating Your Classes.

The following is an example for how to generate Javascript bindings.

```
protoc --proto_path=./ --js_out=import_style=commonjs,binary:src \
opentelemetry/proto/common/v1/common.proto \
opentelemetry/proto/resource/v1/resource.proto \
opentelemetry/proto/metrics/v1/metrics.proto \
opentelemetry/proto/collector/metrics/v1/metrics_service.proto
```

The following section includes examples of using the language-specific bindings that you can build using the previous instructions.

Java

```
package com.example;

import io.opentelemetry.proto.collector.metrics.v1.ExportMetricsServiceRequest;

import java.io.IOException;
import java.io.InputStream;
import java.util.ArrayList;
import java.util.List;

public class MyOpenTelemetryParser {

    public List<ExportMetricsServiceRequest> parse(InputStream inputStream) throws
 IOException {
        List<ExportMetricsServiceRequest> result = new ArrayList<>();

        ExportMetricsServiceRequest request;
        /* A Kinesis record can contain multiple `ExportMetricsServiceRequest`
           records, each of them starting with a header with an
           UnsignedVarInt32 indicating the record length in bytes:
            ------ --------------------------- ------ -----------------------
           |UINT32|ExportMetricsServiceRequest|UINT32|pExportMetricsService...
            ------ --------------------------- ------ -----------------------
         */
        while ((request =
 ExportMetricsServiceRequest.parseDelimitedFrom(inputStream)) != null) {
            // Do whatever we want with the parsed message
```

```
            result.add(request);
        }

        return result;
    }
}
```

Javascript

This example assumes that the root folder with the bindings generated is `./`

The data argument of the function `parseRecord` can be one of the following types:

- `Uint8Array` this is optimal
- `Buffer` optimal under node
- `Array.number` 8-bit integers

```
const pb = require('google-protobuf')
const pbMetrics =
    require('./opentelemetry/proto/collector/metrics/v1/metrics_service_pb')

function parseRecord(data) {
    const result = []

    // Loop until we've read all the data from the buffer
    while (data.length) {
        /* A Kinesis record can contain multiple `ExportMetricsServiceRequest`
           records, each of them starting with a header with an
           UnsignedVarInt32 indicating the record length in bytes:
            ------ -------------------------- ------ -----------------------
           |UINT32|ExportMetricsServiceRequest|UINT32|ExportMetricsService...
            ------ -------------------------- ------ -----------------------
         */
        const reader = new pb.BinaryReader(data)
        const messageLength = reader.decoder_.readUnsignedVarint32()
        const messageFrom = reader.decoder_.cursor_
        const messageTo = messageFrom + messageLength

        // Extract the current `ExportMetricsServiceRequest` message to parse
        const message = data.subarray(messageFrom, messageTo)

        // Parse the current message using the ProtoBuf library
```

```
        const parsed =
            pbMetrics.ExportMetricsServiceRequest.deserializeBinary(message)

        // Do whatever we want with the parsed message
        result.push(parsed.toObject())

        // Shrink the remaining buffer, removing the already parsed data
        data = data.subarray(messageTo)
    }

    return result
}
```

Python

You must read the `var-int` delimiters yourself or use the internal methods `_VarintBytes(size)` and `_DecodeVarint32(buffer, position)`. These return the position in the buffer just after the size bytes. The read-side constructs a new buffer that is limited to reading only the bytes of the message.

```
size = my_metric.ByteSize()
f.write(_VarintBytes(size))
f.write(my_metric.SerializeToString())
msg_len, new_pos = _DecodeVarint32(buf, 0)
msg_buf = buf[new_pos:new_pos+msg_len]
request = metrics_service_pb.ExportMetricsServiceRequest()
request.ParseFromString(msg_buf)
```

Go

Use `Buffer.DecodeMessage()`.

C#

Use `CodedInputStream`. This class can read size-delimited messages.

C++

The functions described in `google/protobuf/util/delimited_message_util.h` can read size-delimited messages.

Other languages

For other languages, see Download Protocol Buffers.

When implementing the parser, consider that a Kinesis record can contain multiple `ExportMetricsServiceRequest` Protocol Buffers messages, each of them starting with a header with an `UnsignedVarInt32` that indicates the record length in bytes.

Troubleshooting metric streams in CloudWatch

If you're not seeing metric data at your final destination, check the following:

- Check that the metric stream is in the running state. For steps on how to use the CloudWatch console to do this, see Metric stream operation and maintenance.
- Metrics published more than two days in the past are not streamed. To determine whether a particular metric will be streamed, graph the metric in the CloudWatch console and check how old the last visible datapoint is. If it is more than two days in the past, then it won't be picked up by metric streams.
- Check the metrics emitted by the metric stream. In the CloudWatch console, under **Metrics**, look at the **AWS/CloudWatch/MetricStreams** namespace for the **MetricUpdate**, **TotalMetricUpdate**, and **PublishErrorRate** metrics.
- If the **PublishErrorRate** metric is high, confirm that the destination that is used by the Firehose delivery stream exists and that the IAM role specified in the metric stream's configuration grants the `CloudWatch` service principal permissions to write to it. For more information, see Trust between CloudWatch and Firehose.
- Check that the Firehose delivery stream has permission to write to the final destination.
- In the Firehose console, view the Firehose delivery stream that is used for the metric stream and check the **Monitoring** tab to see whether the Firehose delivery stream is receiving data.
- Confirm that you have configured your Firehose delivery stream with the correct details.
- Check any available logs or metrics for the final destination that the Firehose delivery stream writes to.
- To get more detailed information, enable CloudWatch Logs error logging on the Firehose delivery stream. For more information, see Monitoring Amazon Data Firehose Using CloudWatch Logs.

> **Note**
>
> After a data point for a specific metric and timestamp has been sent, it won't be sent again even if the value of the data point changes later.

View available metrics

Metrics are grouped first by namespace, and then by the various dimension combinations within each namespace. For example, you can view all EC2 metrics, EC2 metrics grouped by instance, or EC2 metrics grouped by Auto Scaling group.

Only the Amazon services that you're using send metrics to Amazon CloudWatch.

For a list of Amazon services that send metrics to CloudWatch, see Amazon services that publish CloudWatch metrics. From this page, you can also see the metrics and dimensions that are published by each of those services.

> **Note**
>
> Metrics that have not had any new data points in the past two weeks do not appear in the console. They also do not appear when you type their metric name or dimension names in the search box in the **All metrics** tab in the console, and they are not returned in the results of a list-metrics command. The best way to retrieve these metrics is with the get-metric-data or get-metric-statistics commands in the Amazon CLI.
> If the old metric you want to view has a current metric with similar dimensions, you can view that current similar metric and then choose the **Source** tab, and change the metric name and dimension fields to the ones that you want, and also change the time range to a time when the metric was being reported.

The following steps help you browse through the metric namespaces to find and view metrics. You can also search for metrics using targeted search terms. For more information, see Search for available metrics.

If you are browsing in an account set up as a monitoring account in CloudWatch cross-account observability, you can view metrics from the source accounts linked to this monitoring account. When metrics from source accounts are displayed, the ID or label of the account that they are from is also displayed. For more information, see CloudWatch cross-account observability.

To view available metrics by namespace and dimension using the console

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Metrics**, **All metrics**.
3. Select a metric namespace (for example, **EC2** or **Lambda**).
4. Select a metric dimension (for example, **Per-Instance Metrics** or **By Function Name**).
5. The **Browse** tab displays all metrics for that dimension in the namespace. By each metric name is an information button you can choose to see a pop-up with the metric definition.

   If this is a monitoring account in CloudWatch cross-account observability, you also see the metrics from the source accounts linked to this monitoring account. The **Account label** and **Account id** columns in the table display which account each metric is from.

   You can do the following:

   a. To sort the table, use the column heading.
   b. To graph a metric, select the check box next to the metric. To select all metrics, select the check box in the heading row of the table.
   c. To filter by account, choose the account label or account ID and then choose **Add to search**.
   d. To filter by resource, choose the resource ID and then choose **Add to search**.
   e. To filter by metric, choose the metric name and then choose **Add to search**.
6. (Optional) To add this graph to a CloudWatch dashboard, choose **Actions**, **Add to dashboard**.

To view available metrics by account namespace, dimension, or metric using the Amazon CLI

Use the list-metrics command to list CloudWatch metrics. For a list of the namespaces, metrics, and dimensions for all services that publish metrics, see Amazon services that publish CloudWatch metrics.

The following example command lists all the metrics for Amazon EC2.

```
aws cloudwatch list-metrics --namespace AWS/EC2
```

The following is example output.

```
{
```

```
  "Metrics" : [
    ...
    {
        "Namespace": "AWS/EC2",
        "Dimensions": [
            {
                "Name": "InstanceId",
                "Value": "i-1234567890abcdef0"
            }
        ],
        "MetricName": "NetworkOut"
    },
    {
        "Namespace": "AWS/EC2",
        "Dimensions": [
            {
                "Name": "InstanceId",
                "Value": "i-1234567890abcdef0"
            }
        ],
        "MetricName": "CPUUtilization"
    },
    {
        "Namespace": "AWS/EC2",
        "Dimensions": [
            {
                "Name": "InstanceId",
                "Value": "i-1234567890abcdef0"
            }
        ],
        "MetricName": "NetworkIn"
    },
    ...
  ]
}
```

To list all the available metrics for a specified resource

The following example specifies the `AWS/EC2` namespace and the `InstanceId` dimension to view the results for the specified instance only.

```
aws cloudwatch list-metrics --namespace AWS/EC2 --dimensions
 Name=InstanceId,Value=i-1234567890abcdef0
```

To list a metric for all resources

The following example specifies the AWS/EC2 namespace and a metric name to view the results for the specified metric only.

```
aws cloudwatch list-metrics --namespace AWS/EC2 --metric-name CPUUtilization
```

To retrieve metrics from linked source accounts in CloudWatch cross-account observability

The following example is run in a monitoring account to retrieve metrics from both the monitoring account and all linked source accounts. If you do not add `--include-linked-accounts`, the command returns only the monitoring account's metrics.

```
aws cloudwatch list-metrics --include-linked-accounts
```

To retrieve metrics from a source account in CloudWatch cross-account observability

The following example is run in a monitoring account to retrieve metrics from the source account with the ID 111122223333.

```
aws cloudwatch list-metrics --include-linked-accounts --owning-account "111122223333"
```

Search for available metrics

You can search within all of the metrics in your account using targeted search terms. Metrics are returned that have matching results within their namespace, metric name, or dimensions.

If this is a monitoring account in CloudWatch cross-account observability, you also search for metrics from the source accounts linked to this monitoring account.

> **Note**
>
> Metrics that have not had any new data points in the past two weeks do not appear in the console. They also do not appear when you type their metric name or dimension names in the search box in the **All metrics** tab in the console, and they are not returned in the results of a list-metrics command. The best way to retrieve these metrics is with the get-metric-data or get-metric-statistics commands in the Amazon CLI.

To search for available metrics in CloudWatch

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Metrics**, **All metrics**.
3. In the search field, enter a search term, such as a metric name, namespace, account ID, account label, dimension name or value, or resource name. This shows you all of the namespaces with metrics with this search term.

   For example, if you search for **`volume`**, this shows the namespaces that contain metrics with this term in their name.

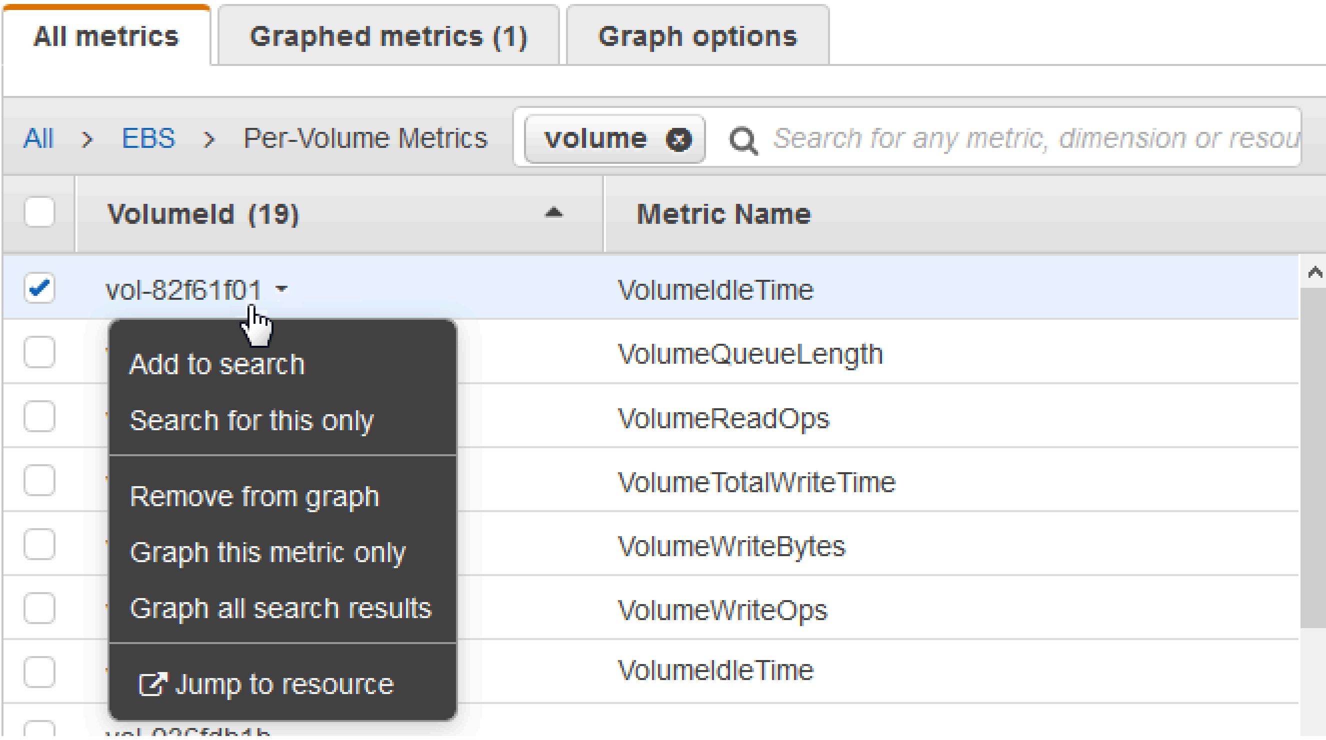
   For more information on search, see Use search expressions in graphs
4. To graph all the search results, choose **Graph search**

   or

   Select a namespace to view the metrics from that namespace. You can then do the following:

   a. To graph one or more metrics, select the check box next to each metric. To select all metrics, select the check box in the heading row of the table.
   b. To refine your search, hover over a metric name and choose **Add to search** or **Search for this only**.
   c. To view one of the resources on its console, choose the resource ID and then choose **Jump to resource**.
   d. To view help for a metric, select the metric name and choose **What is this?**.

   The selected metrics appear on the graph.



5. (Optional) Select one of the buttons in the search bar to edit that part of the search term.

Graphing metrics

Use the CloudWatch console to graph metric data generated by other Amazon services. This makes it more efficient to see the metric activity on your services. The following procedures describe how to graph metrics in CloudWatch.

Contents

Graph a metric

You can select metrics and create graphs of the metric data using the CloudWatch console.

CloudWatch supports the following statistics on metrics: `Average`, `Minimum`, `Maximum`, `Sum`, and `SampleCount`. For more information, see Statistics.

You can view your data at different levels of detail. For example, you can choose a one-minute view, which can be useful when troubleshooting. Or, choose a less detailed, one-hour view. That can be useful when viewing a broader time range (for example, 3 days) so that you can see trends over time. For more information, see Periods.

If you are using an account that is set up as a monitoring account in CloudWatch cross-account observability, you can graph metrics from the source accounts linked to this monitoring account. For more information, see CloudWatch cross-account observability.

Creating a graph

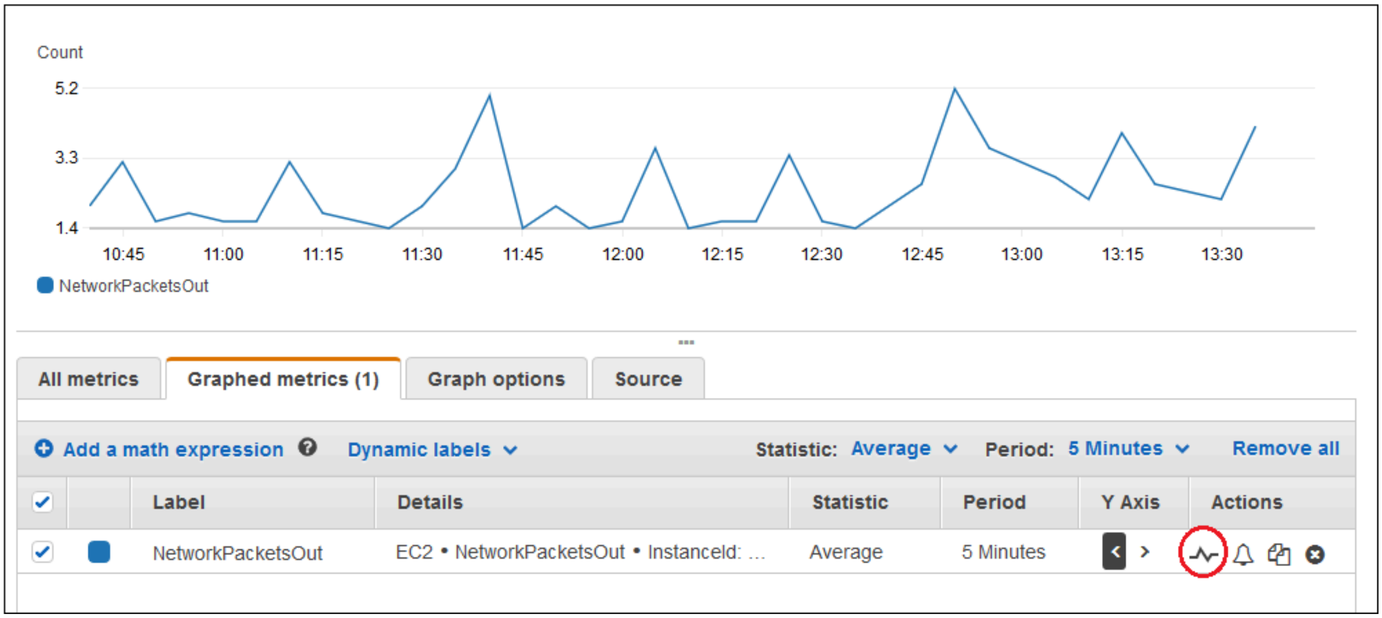
To graph a metric

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Metrics**, **All metrics**.
3. In the **Browse** tab, enter a search term in the search field, such as a metric name, account ID, or resource name.

   For example, if you search for the `CPUUtilization` metric, you see the namespaces and dimensions with this metric.
4. Select one of the results for your search to view the metrics.
5. To graph one or more metrics, select the check box next to each metric. To select all metrics, select the check box in the heading row of the table.
6. (Optional) To change the type of graph, choose the **Options** tab. You can then choose between a line graph, stacked area chart, number display, gauge, bar chart, or pie chart.
7. Choose the **Graphed metrics** tab.
8. (Optional) To change the statistic used in the graph, choose the new statistic in the **Statistic** column next to the metric name.

   For more information about CloudWatch statistics, see CloudWatch statistics definitions. For more information about the **p*xx* percentile statistics, see Percentiles.**

9. (Optional) To add an anomaly detection band that shows expected values for the metric, choose the anomaly detection icon under **Actions** next to the metric. For more information about anomaly detection, see Using CloudWatch anomaly detection.



CloudWatch uses up to two weeks of the metric's recent historical data to calculate a model for expected values. It then displays this range of expected values as a band on the graph. CloudWatch adds a new row under the metric to display the anomaly detection band math expression, labeled **ANOMALY_DETECTION_BAND**. If recent historical data exists, you immediately see a preview anomaly detection band, which is an approximation of the anomaly detection band generated by the model. It takes up to 15 minutes for the actual anomaly detection band to appear.

By default, CloudWatch creates the upper and lower bounds of the band of expected values with a default value of 2 for the band threshold. To change this number, change the value at the end of the formula under **Details** for the band.

- (Optional) Choose **Edit model** to change how the anomaly detection model is calculated. You can exclude past and future time periods from being used in the training for calculating the model. It is critical to exclude unusual events system as system outage, deployments, and holidays from the training data. You can also specify the time zone to use for the model for daylight saving time changes.

  For more information, see Editing an anomaly detection model .

For more information about anomaly detection, see Using CloudWatch anomaly detection.

To hide the model from the graph, remove the check mark from the line with the `ANOMALY_DETECTION_BAND` function or choose the `X` icon. To delete the model entirely, choose **Edit model**, **Delete model**.

10. (Optional) As you choose metrics to graph, specify a dynamic label to appear on the graph legend for each metric. Dynamic labels display a statistic about the metric, and automatically update when the dashboard or graph is refreshed. To add a dynamic label, choose **Graphed metrics**, **Add dynamic label**.

    By default, the dynamic values that you add to the label appear at the beginning of the label. You can then choose the **Label** value for the metric to edit the label. For more information, see Use dynamic labels.

11. To view more information about the metric being graphed, pause the mouse over the legend.

12. Horizontal annotations can help graph users more efficiently see when a metric has spiked to a certain level, or whether the metric is within a predefined range. To add a horizontal annotation, choose the **Options** tab and then **Add horizontal annotation**:

    a. For **Label**, enter a label for the annotation.

    b. For **Value**, enter the metric value where the horizontal annotation appears.

    c. For **Fill**, specify whether to use fill shading with this annotation. For example, choose `Above` or `Below` for the corresponding area to be filled. If you specify `Between`, another `Value` field appears, and the area of the graph between the two values is filled.

    d. For **Axis**, specify whether the numbers in `Value` refer to the metric associated with the left Y-axis or the right Y-axis, if the graph includes multiple metrics.

       You can change the fill color of an annotation by choosing the color square in the left column of the annotation.

    Repeat these steps to add multiple horizontal annotations to the same graph.

    To hide an annotation, clear the check box in the left column for that annotation.

    To delete an annotation, choose **x** in the **Actions** column.

13. To get a URL for your graph, choose **Actions**, **Share**. Copy the URL to save or share.

14. To add your graph to a dashboard, choose **Actions**, **Add to dashboard**.

Creating a graph of metrics from another data source

You can create a graph that displays resources from data sources other than CloudWatch. For more information about creating connections to these other data sources, see Query metrics from other data sources.

To graph a metric from another data source

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Metrics**, **All metrics**.
3. Choose the **Multi source query** tab.
4. For **Data source**, select the data source that you want to use.

   If you haven't already created a connection to the data source that you want, select **Create and manage data sources**, then choose **Create and manage data sources**. For information about the rest of this data source creation process, see Connect to a prebuilt data source with a wizard.
5. The wizard or query editor prompts you for the information necessary for the query. The workflow is different for each data source, and is tailored to the data source. For example, for Amazon Managed Service for Prometheus; and Prometheus data sources, a PromQL query editor box with a query helper appear.
6. When you have finished constructing the query, choose **Graph query**.

   The graph is populated with metrics from the query.
7. (Optional) Horizontal annotations can help graph users more efficiently see when a metric has spiked to a certain level, or whether the metric is within a predefined range. To add a horizontal annotation, choose the **Options** tab and then **Add horizontal annotation**:

   a. For **Label**, enter a label for the annotation.

   b. For **Value**, enter the metric value where the horizontal annotation appears.

   c. For **Fill**, specify whether to use fill shading with this annotation. For example, choose `Above` or `Below` for the corresponding area to be filled. If you specify `Between`, another `Value` field appears, and the area of the graph between the two values is filled.

   d. For **Axis**, specify whether the numbers in `Value` refer to the metric associated with the left Y-axis or the right Y-axis, if the graph includes multiple metrics.

      You can change the fill color of an annotation by choosing the color square in the left column of the annotation.

   Repeat these steps to add multiple horizontal annotations to the same graph.

   To hide an annotation, clear the check box in the left column for that annotation.

   To delete an annotation, choose **x** in the **Actions** column.

8. (Optional) To add this graph to a dashboard, choose **Actions**, **Add to dashboard**.

Updating a graph

To update your graph

1. To change the name of the graph, choose the pencil icon.
2. To change the time range, select one of the predefined values or choose **custom**. For more information, see Modify the time range or time zone format for a graph.
3. To change the statistic, choose the **Graphed metrics** tab. Choose the column heading or an individual value and then choose one of the statistics or predefined percentiles, or specify a custom percentile (for example, **`p95.45`**).
4. To change the period, choose the **Graphed metrics** tab. Choose the column heading or an individual value and then choose a different value.
5. To add a horizontal annotation, choose **Graph options** and then **Add horizontal annotation**:

   a. For **Label**, enter a label for the annotation.

   b. For **Value**, enter the metric value where the horizontal annotation appears.

   c. For **Fill**, specify whether to use fill shading with this annotation. For example, choose `Above` or `Below` for the corresponding area to be filled. If you specify `Between`, another `Value` field appears, and the area of the graph between the two values is filled.

   d. For **Axis**, specify whether the numbers in `Value` refer to the metric associated with the left y-axis or the right y-axis, if the graph includes multiple metrics.

You can change the fill color of an annotation by choosing the color square in the left column of the annotation.

Repeat these steps to add multiple horizontal annotations to the same graph.

To hide an annotation, clear the check box in the left column for that annotation.

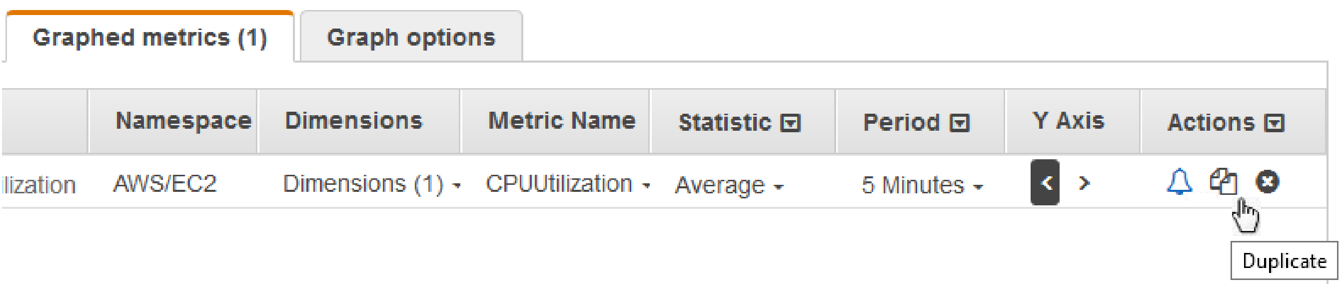
To delete an annotation, choose **x** in the **Actions** column.

6. To change the refresh interval, choose **Refresh options** and then select **Auto refresh** or choose **1 Minute**, **2 Minutes**, **5 Minutes**, or **15 Minutes**.

Duplicating a metric

To duplicate a metric

1. Choose the **Graphed metrics** tab.
2. For **Actions**, choose the **Duplicate** icon.



3. Update the duplicate metric as needed.

Merge two graphs into one

You can merge two different graphs into one, and then the resulting graph shows both metrics. This can be useful if you already have different metrics displayed in different graphs and want to combine them, or you want to easily create a single graph with metrics from different Regions.

To merge a graph into another one, you use either the URL or JSON source of the graph that you want to merge in.

To merge two graphs into one

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. Open the graph that you want to merge into another graph. To do so, you can choose **Metrics**, **All metrics**, and then choose a metric to graph. Or you can open a dashboard and then open one of the graphs on the dashboard by selecting the graph and choosing **Open in metrics** from the menu at the upper right of the graph.
3. After you have a graph open, do one of the following:.
   - Copy the URL from the browser bar.
   - Choose the **Source** tab and then choose **Copy**.
4. Open the graph that you want to merge the previous graph into.
5. When you have the second graph open in the **Metrics** view, choose **Actions**, **Merge graph**.
6. Enter the URL or JSON that you previously copied, and choose **Merge**.
7. The merged graphs appear. The y-axis on the left is for the original graph, and the y-axis on the right is for the graph that you merged into it.

   > **Note**
   >
   > If the graph that you merged into uses the **METRICS()** function, the metrics in the graph that was merged in are not included in the **METRICS()** calculation in the merged graph.
8. To save the merged graph to a dashboard, choose **Actions**, **Add to dashboard**.

Use dynamic labels

You can use dynamic labels with your graphs. Dynamic labels add a dynamically updated value to the label for the selected metric. You can add a wide range of values to the labels, as shown in the following tables.

The dynamic value shown in the label is derived from the time range currently shown on the graph. The dynamic part of the label automatically updates when either the dashboard or the graph is refreshed.

If you use a dynamic label with a search expression, the dynamic label applies to every metric returned by the search.

You can use the CloudWatch console to add a dynamic value to a label, edit the label, change the position of the dynamic value within the label column, and make other customizations.

Dynamic labels

Within a dynamic label, you can use the following values relating to properties of the metric:

Dynamic label live value	Description
${AVG}	The average of the values in the time range currently shown in the graph.
${DATAPOINT_COUNT}	The number of data points in the time range that is currently shown in the graph.
${FIRST}	The oldest of the metric values in the time range that is currently shown in the graph.
${FIRST_LAST_RANGE}	The difference between the metric values of the oldest and newest data points that are currently shown in the graph.
${FIRST_LAST_TIME_RANGE}	The absolute time range between the oldest and newest data points that are currently shown in the graph.
${FIRST_TIME}	The timestamp of the oldest data point in the time range that is currently shown in the graph.
${FIRST_TIME_RELATIVE}	The absolute time difference between now and the timestamp of the oldest data point in the time range that is currently shown in the graph.
${LABEL}	The representation of the default label for a metric.
${LAST}	The most recent of the metric values in the time range that is currently shown in the graph.
${LAST_TIME}	The timestamp of the newest data point in the time range that is currently shown in the graph.

Dynamic label live value	Description
${LAST_TIME_RELATIVE}	The absolute time difference between now and the timestamp of the newest data point in the time range that is currently shown in the graph.
${MAX}	The maximum of the values in the time range currently shown in the graph.
${MAX_TIME}	The timestamp of the data point that has the highest metric value, of the data points that are currently shown in the graph.
${MAX_TIME_RELATIVE}	The absolute time difference between now and the timestamp of the data point with the highest value, of those data points that are currently shown in the graph.
${MIN}	The minimum of the values in the time range currently shown in the graph.
${MIN_MAX_RANGE}	The difference in metric values between the data points with the highest and lowest metric values, of those data points that are currently shown in the graph.
${MIN_MAX_TIME_RANGE}	The absolute time range between the data points with the highest and lowest metric values, of those data points that are currently shown in the graph.
${MIN_TIME}	The timestamp of the data point that has the lowest metric value, of the data points that are currently shown in the graph.
${MIN_TIME_RELATIVE}	The absolute time difference between now and the timestamp of the data point with the lowest value, of those data points that are currently shown in the graph.
${PROP('AccountId')}	The Amazon account ID of the metric.
${PROP('AccountLabel')}	The label specified for the source account that owns this metric, in CloudWatch cross-account observability.

Dynamic label live value	Description
${PROP('Dim.*dimension_name*')}	The value of the specified dimension. Replace *dimension_name* with the case-sensitive name of your dimension.
${PROP('MetricName')}	The name of the metric.
${PROP('Namespace')}	The namespace of the metric.
${PROP('Period')}	The period of the metric, in seconds.
${PROP('Region')}	The Amazon Region where the metric is published.
${PROP('Stat')}	The metric statistic that is being graphed.
${SUM}	The sum of the values in the time range currently shown in the graph.

For example, suppose you have a search expression **`SEARCH(' {AWS/Lambda, FunctionName} Errors ', 'Sum')`**, which finds the `Errors` for each of your Lambda functions. If you set the label to be `[max: ${MAX} Errors for Function Name ${LABEL}]`, the label for each metric is **[max: *number* Errors for Function Name *Name*]**.

You can add as many as six dynamic values to a label. You can use the `${LABEL}` placeholder only once within each label.

Modify the time range or time zone format for a graph

This section describes how you can modify the date, time, and time zone format on a CloudWatch metrics graph. It also describes how you can zoom in on a graph to apply a specific time range. For information about creating a graph, see Graph a metric.

> **Note**
>
> If the time range of a dashboard is shorter than the period used for a graph on the dashboard, the following happens:

- The graph is modified to display the amount of data corresponding one complete period for that widget, even though this is longer than the dashboard time range. This ensures that there is at least one data point on the graph.
- The start time of the period for this data point is adjusted backwards to ensure that at least one data point can be displayed.

Set a relative time range

New interface

To specify a relative time range for a graph

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Metrics**, and then choose **All metrics**. In the upper right corner of the screen, you can select one of the predefined time ranges, which span from 1 hour to 1 week (**1h**, **3h**, **12h**, **1d**, **3d**, or **1w**). Alternatively, you can choose **Custom** to set your own time range.
3. Choose **Custom**, and then select the **Relative** tab in the upper left corner of the box. You can specify a time range in **Minutes**, **Hours**, **Days**, **Weeks**, **Months**.
4. After you specify a time range, choose **Apply**.

Original interface

To specify a relative time range for a graph

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Metrics**, and then choose **All metrics**. In the upper right corner of the screen, you can select one of the predefined time ranges, which span from 1 hour to 1 week (**1h**, **3h**, **12h**, **1d**, **3d**, or **1w**). Alternatively, you can choose **custom** to set your own time range.
3. Choose **custom**, and then choose **Relative** in the upper left corner of the box. You can specify a time range in **Minutes**, **Hours**, **Days**, **Weeks**, or **Months**.

Set an absolute time range

New interface

To specify an absolute time range for a graph

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Metrics**, and then choose **All metrics**. In the upper right corner of the screen, you can select one of the predefined time ranges, which span from 1 hour to 1 week (**1h**, **3h**, **12h**, **1d**, **3d**, or **1w**). Alternatively, you can choose **Custom** to set your own time range.
3. Choose **Custom**, and then select the **Absolute** tab in the upper left corner of the box. Use the calendar picker or text field boxes to specify a time range.
4. After you specify a time range, choose **Apply**.

Original interface

To specify an absolute time range for a graph

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Metrics**, and then choose **All metrics**. In the upper right corner of the screen, you can select one of the predefined time ranges, which span from 1 hour to 1 week (**1h**, **3h**, **12h**, **1d**, **3d**, or **1w**). Alternatively, you can choose **custom** to set your own time range.
3. Choose **custom**, and then choose **Absolute** in the upper left corner of the box. Use the calendar picker or text field boxes to specify a time range.
4. After you specify a time range, choose **Apply**.

Set the time zone format

New interface

To specify the time zone for a graph

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Metrics**, and then choose **All metrics**. In the upper right corner of the screen, you can select one of the predefined time ranges, which span from 1

hour to 1 week (**1h**, **3h**, **12h**, **1d**, **3d**, or **1w**). Alternatively, you can choose **Custom** to set your own time range.

3. Choose **Custom**, and then choose the dropdown in the upper right corner of the box. You can change the time zone to **UTC** or **Local time zone**.
4. After you make your changes, choose **Apply**.

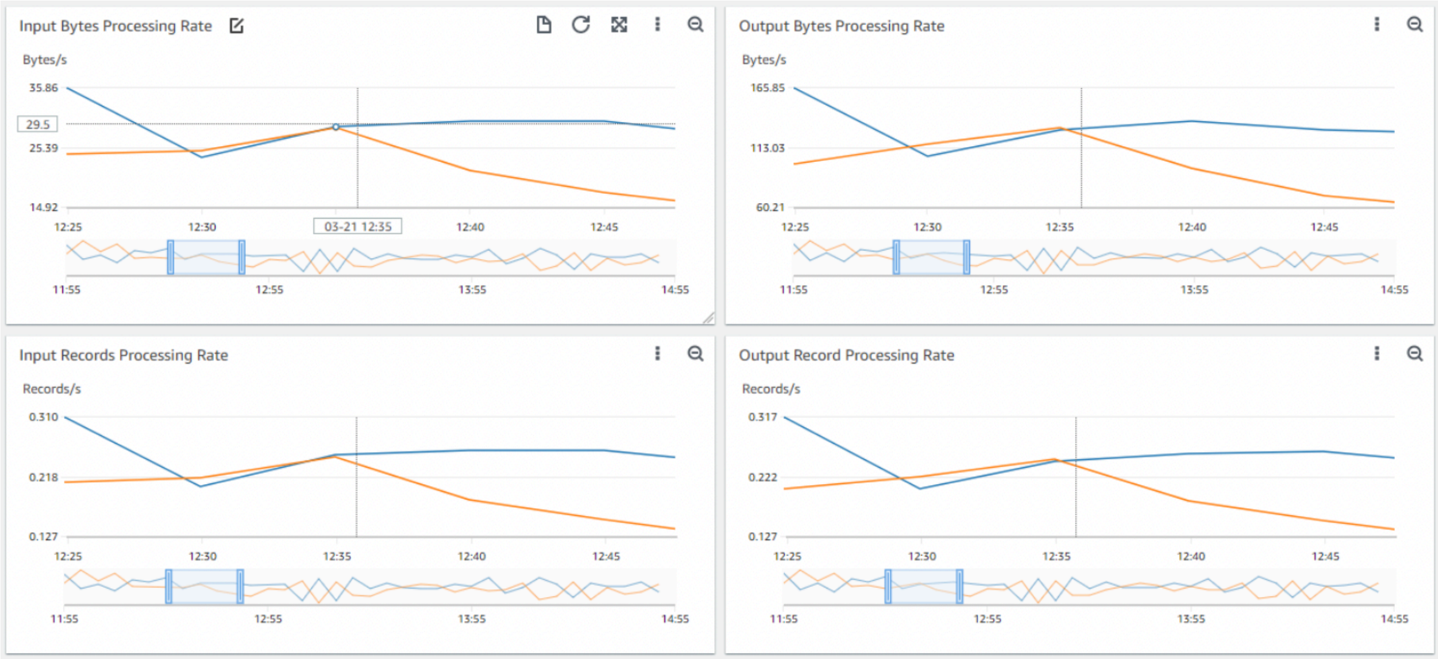
Original interface

To specify the time zone for a graph

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Metrics**, and then choose **All metrics**. In the upper right corner of the screen, you can select one of the predefined time ranges, which span from 1 hour to 1 week (**1h**, **3h**, **12h**, **1d**, **3d**, or **1w**). Alternatively, you can choose **custom** to set your own time range.
3. Choose **custom**, and then choose the dropdown in the upper right corner of the box. You can change the time zone to **UTC** or **Local timezone**.

Zoom in on a line graph or stacked area graph

In the CloudWatch console, you can use the mini-map zoom feature to focus on sections of line graphs and stacked area graphs without changing between zoomed-in and zoomed-out views. For example, you can use the mini-map zoom feature to focus on a peak in a line graph, so that you can compare the spike against other metrics in your dashboard from the same timeline. The procedures in this section describe how to use the zoom feature.



In the preceding image, the zoom feature focuses on a spike in a line graph that's related to the input bytes processing rate while also showing other line graphs in the dashboard that focus on sections from the same timeline.

New interface

To zoom in on a graph

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Metrics**, and then choose **All metrics**.
3. Choose **Browse**, and then select a metric or metrics to graph.
4. Choose **Options**, and select **Line** under ***Widget type***.
5. Choose and drag on the area of the graph that you want to focus on, and then release the drag.
6. To reset the zoom, choose the **Reset zoom** icon, which looks like a magnifying glass with a minus (-) symbol inside.

Original interface

To zoom in on a graph

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.

2. In the navigation pane, choose **Metrics**, and then choose **All metrics**. .
3. Choose **All metrics**, and then select a metric to graph.
4. Choose **Graph options**. Under ***Widget type*** select **Line**.
5. Choose and drag on the area of the graph that you want to focus on, and then release the drag.
6. To reset the zoom, choose the **Reset zoom** icon, which looks like a magnifying glass with a minus (-) symbol inside.

> **Tip**
>
> If you already created a dashboard that contains a line graph or stacked area graph, you can go to the dashboard and begin using the zoom feature.

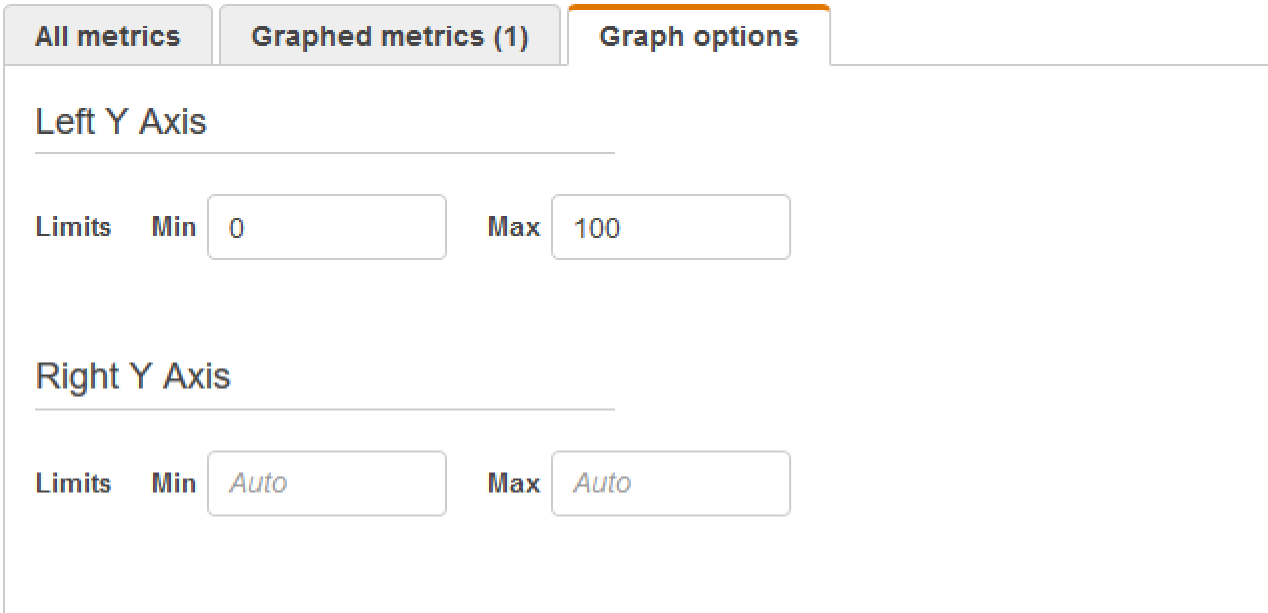
Modify the y-axis for a graph

You can set custom bounds for the y-axis on a graph to help you see the data better. For example, you can change the bounds on a `CPUUtilization` graph to 100 percent so that it's easy to see whether the CPU is low (the plotted line is near the bottom of the graph) or high (the plotted line is near the top of the graph).

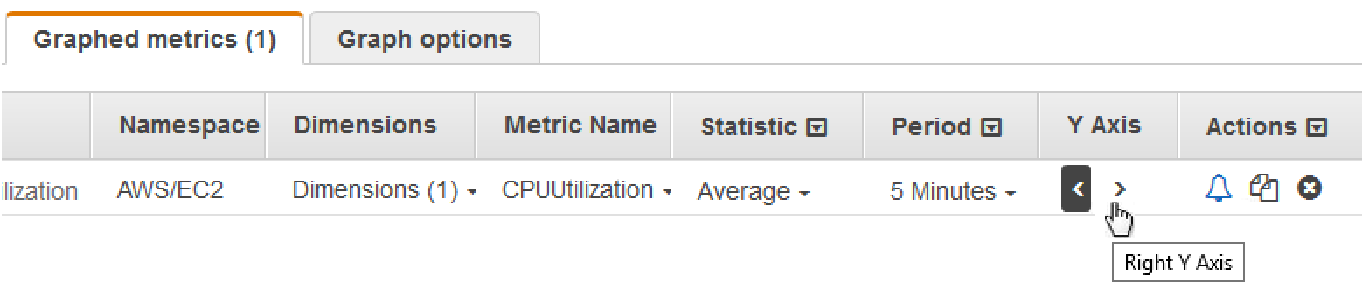
You can switch between two different y-axes for your graph. This is useful if the graph contains metrics that have different units or that differ greatly in their range of values.

To modify the y-axis on a graph

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Metrics**, **All metrics**.
3. Select a metric namespace (for example, **EC2**) and then a metric dimension (for example, **Per-Instance Metrics**).
4. The **All metrics** tab displays all metrics for that dimension in that namespace. To graph a metric, select the check box next to the metric.
5. On the **Graph options** tab, specify the **Min** and **Max** values for **Left Y Axis**. The value of **Min** can't be greater than the value of **Max**.



6. To create a second y-axis, specify the **Min** and **Max** values for **Right Y Axis**.
7. To switch between the two y-axes, choose the **Graphed metrics** tab. For **Y Axis**, choose **Left Y Axis** or **Right Y Axis**.



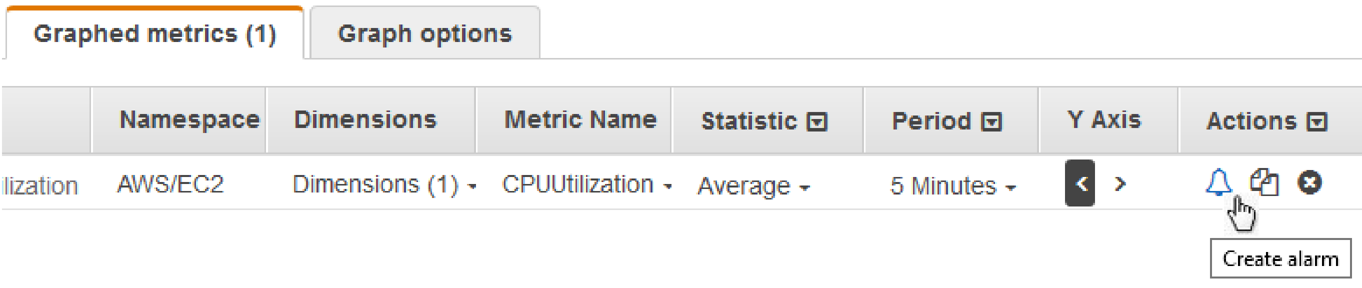
Create an alarm from a metric on a graph

You can graph a metric and then create an alarm from the metric on the graph, which has the benefit of populating many of the alarm fields for you.

To create an alarm from a metric on a graph

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Metrics**, **All metrics**.
3. Select a metric namespace (for example, **EC2**) and then a metric dimension (for example, **Per-Instance Metrics**).

4. The **All metrics** tab displays all metrics for that dimension in that namespace. To graph a metric, select the check box next to the metric.
5. To create an alarm for the metric, choose the **Graphed metrics** tab. For **Actions**, choose the alarm icon.



6. Under **Conditions**, choose **Static** or **Anomaly detection** to specify whether to use a static threshold or anomaly detection model for the alarm.

   Depending on your choice, enter the rest of the data for the alarm conditions.
7. Choose **Additional configuration**. For **Datapoints to alarm**, specify how many evaluation periods (data points) must be in the `ALARM` state to trigger the alarm. If the two values here match, you create an alarm that goes to `ALARM` state if that many consecutive periods are breaching.

   To create an M out of N alarm, specify a lower number for the first value than you specify for the second value. For more information, see Evaluating an alarm.
8. For **Missing data treatment**, choose how to have the alarm behave when some data points are missing. For more information, see Configuring how CloudWatch alarms treat missing data.
9. Choose **Next**.
10. Under **Notification**, select an SNS topic to notify when the alarm is in `ALARM` state, `OK` state, or `INSUFFICIENT_DATA` state.

    To have the alarm send multiple notifications for the same alarm state or for different alarm states, choose **Add notification**.

    To have the alarm not send notifications, choose **Remove**.
11. To have the alarm perform Auto Scaling or EC2 actions, choose the appropriate button and choose the alarm state and action to perform.
12. When finished, choose **Next**.

13. Enter a name and description for the alarm. The name must contain only ASCII characters. Then choose **Next**.
14. Under **Preview and create**, confirm that the information and conditions are what you want, then choose **Create alarm**.

Using CloudWatch anomaly detection

When you enable *anomaly detection* for a metric, CloudWatch applies statistical and machine learning algorithms. These algorithms continuously analyze metrics of systems and applications, determine normal baselines, and surface anomalies with minimal user intervention.

The algorithms generate an anomaly detection model. The model generates a range of expected values that represent normal metric behavior.

You can enable anomaly detection using the Amazon Web Services Management Console, the Amazon CLI, Amazon CloudFormation, or the Amazon SDK. You can enable anomaly detection on metrics vended by Amazon and also on custom metrics. In an account set up as a monitoring account for CloudWatch cross-account observability, you can create anomaly detectors on metrics in source accounts in addition to metrics in the monitoring account.

You can use the model of expected values in two ways:

- Create anomaly detection alarms based on a metric's expected value. These types of alarms don't have a static threshold for determining alarm state. Instead, they compare the metric's value to the expected value based on the anomaly detection model.

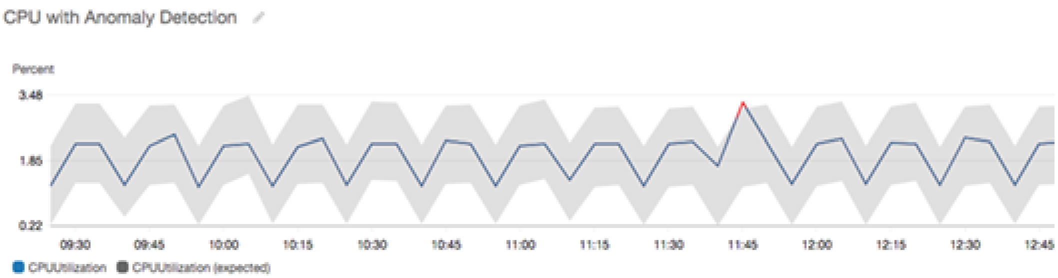
  You can choose whether the alarm is triggered when the metric value is above the band of expected values, below the band, or both.

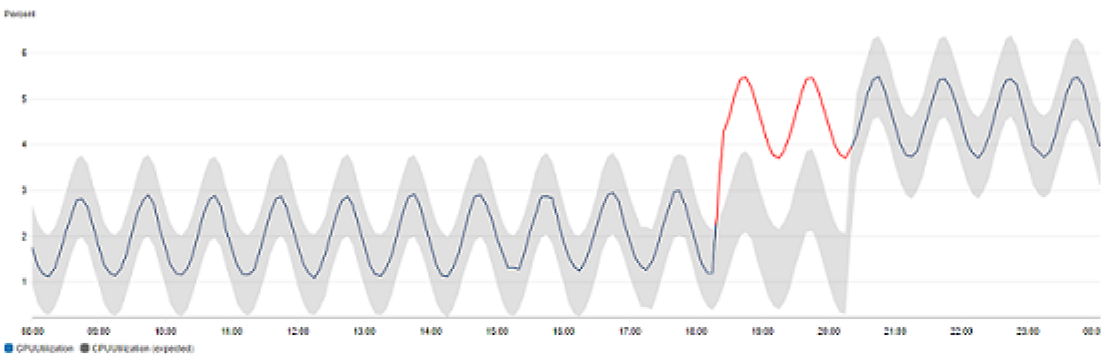
  For more information, see Create a CloudWatch alarm based on anomaly detection.
- When viewing a graph of metric data, overlay the expected values onto the graph as a band. This makes it visually clear which values in the graph are out of the normal range. For more information, see Creating a graph.

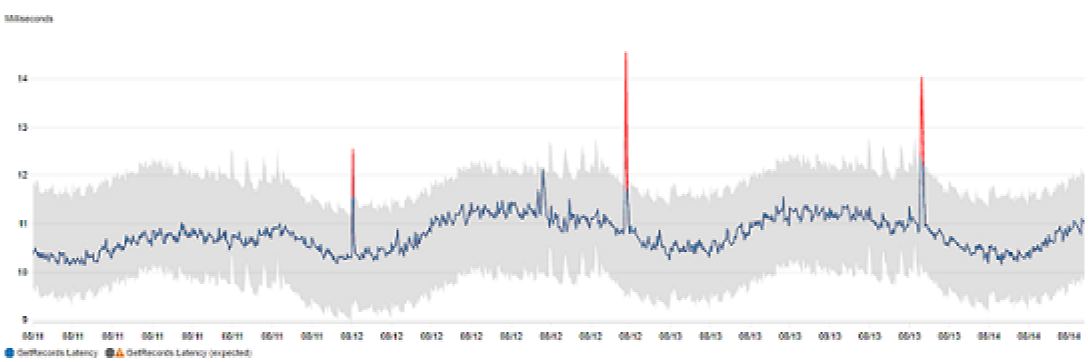
  You can also retrieve the upper and lower values of the model's band by using the `GetMetricData` API request with the `ANOMALY_DETECTION_BAND` metric math function. For more information, see GetMetricData.

In a graph with anomaly detection, the expected range of values is shown as a gray band. If the metric's actual value goes beyond this band, it is shown as red during that time.

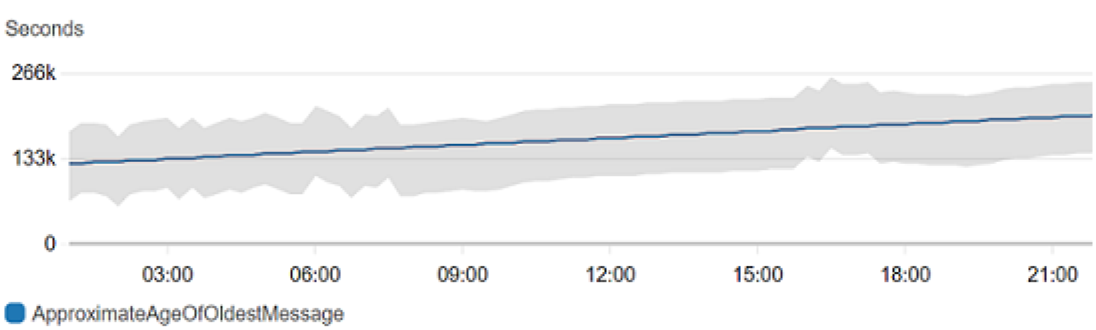
Anomaly detection algorithms account for the seasonality and trend changes of metrics. The seasonality changes could be hourly, daily, or weekly, as shown in the following examples.



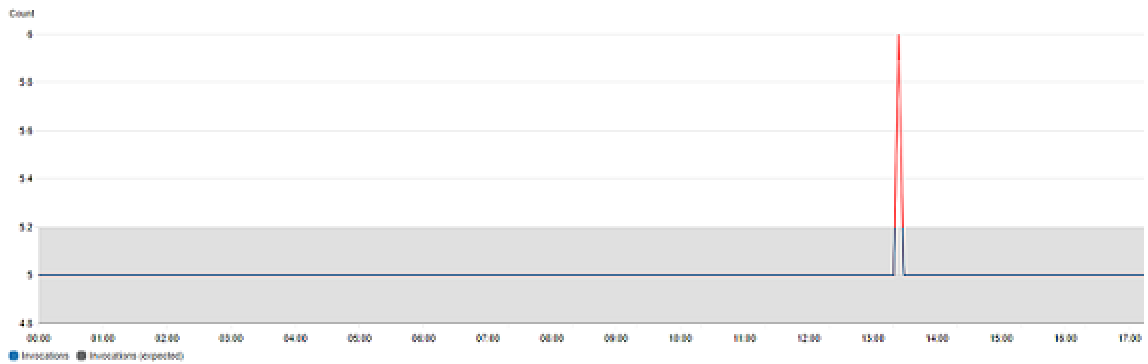




The longer-range trends could be downward or upward.



Anomaly detections also works well with metrics with flat patterns.



How CloudWatch anomaly detection works

When you enable anomaly detection for a metric, CloudWatch applies machine learning algorithms to the metric's past data to create a model of the metric's expected values. The model assesses both trends and hourly, daily, and weekly patterns of the metric. The algorithm trains on up to two weeks of metric data, but you can enable anomaly detection on a metric even if the metric does not have a full two weeks of data.

You specify a value for the anomaly detection threshold that CloudWatch uses along with the model to determine the "normal" range of values for the metric. A higher value for the anomaly detection threshold produces a thicker band of "normal" values.

The machine learning model is specific to a metric and a statistic. For example, if you enable anomaly detection for a metric using the `AVG` statistic, the model is specific to the `AVG` statistic.

When CloudWatch creates a model for many common metrics from Amazon services, it ensures that the band doesn’t extend outside of logical values. For example, the band for `MemoryUtilization` of an EC2 instance will stay between 0 and 100, and the bands tracking CloudFront `Requests`, which can't be negative, will never extend below zero.

After you create a model, CloudWatch anomaly detection continually evaluates the model and makes adjustments to it to ensure that it is as accurate as possible. This includes re-training the model to adjust if the metric values evolve over time or have sudden changes, and also includes predictors to improve the models of metrics that are seasonal, spiky, or sparse.

After you enable anomaly detection on a metric, you can choose to exclude specified time periods of the metric from being used to train the model. This way, you can exclude deployments or other unusual events from being used for model training, ensuring the most accurate model is created.

Using anomaly detection models for alarms incurs charges on your Amazon account. For more information, see [Amazon CloudWatch Pricing](#).

Anomaly detection on metric math

Anomaly detection on metric math is a feature that you can use to create anomaly detection alarms on the output metric math expressions. You can use these expressions to create graphs that visualize anomaly detection bands. The feature supports basic arithmetic functions, comparison and logical operators, and most other functions. For information about functions that are not supported, see Using metric math in the *Amazon CloudWatch User Guide*.

You can create anomaly detection models based on metric math expressions similar to how you already create anomaly detection models. From the CloudWatch console, you can apply anomaly detection to metric math expressions and select anomaly detection as a threshold type for these expressions.

> **Note**
>
> Anomaly detection on metric math only can be enabled and edited in the latest version of the metrics user interface. When you create anomaly detectors based on metric math expressions in the new version of the interface, you can view them in the old version, but not edit them.

For information about how to create, edit, and delete alarms and models for anomaly detection and metric math, see the following sections:

- Create a CloudWatch alarm based on anomaly detection
- Editing an anomaly detection model
- Deleting an anomaly detection model
- Creating a CloudWatch alarm based on a metric math expression

You also can create, delete, and discover anomaly detection models based on metric math expressions using the CloudWatch API with `PutAnomalyDetector`, `DeleteAnomalyDetector`, and `DescribeAnomalyDetectors`. For information about these API actions, see the following sections in the *Amazon CloudWatch API Reference*.

- PutAnomalyDetector

- DeleteAnomalyDetector
- DescribeAnomalyDetectors

For information about how anomaly detection alarms are priced, see Amazon CloudWatch pricing.

Using math expressions with CloudWatch metrics

Metric math enables you to query multiple CloudWatch metrics and use math expressions to create new time series based on these metrics. You can visualize the resulting time series on the CloudWatch console and add them to dashboards. Using Amazon Lambda metrics as an example, you could divide the `Errors` metric by the `Invocations` metric to get an error rate. Then add the resulting time series to a graph on your CloudWatch dashboard.

You can also perform metric math programmatically, using the `GetMetricData` API operation. For more information, see GetMetricData.

Add a math expression to a CloudWatch graph

You can add a math expression to a graph on your CloudWatch dashboard. Each graph is limited to using a maximum of 500 metrics and expressions, so you can add a math expression only if the graph has 499 or fewer metrics. This applies even if not all the metrics are displayed on the graph.

To add a math expression to a graph

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. Create or edit a graph. There needs to be at least one metric in the graph.
3. Choose **Graphed metrics**.
4. Choose **Math expression**, **Start with empty expression**. A new line appears for the expression.
5. In the new line, under the **Details** column, enter the math expression. The tables in the **Metric Math Syntax and Functions** section list the functions that you can use in the expression.

   To use a metric or the result of another expression as part of the formula for this expression, use the value shown in the **Id** column: for example, **m1+m2** or **e1-MIN(e1)**.

   You can change the value of **Id**. It can include numbers, letters, an underscore, and must start with a lowercase letter. Changing the value of **Id** to a more meaningful name can also make a graph easier to understand;for example, changing **m1** and **m2** to **errors** and **requests**.

> **Tip**
>
> Choose the down arrow next to **Math Expression** to see a list of supported functions, which you can use when creating your expression.

6. For the **Label** column of the expression, enter a name that describes what the expression is calculating.

   If the result of an expression is an array of time series, each of those time series is displayed on the graph with a separate line, with different colors. Immediately under the graph is a legend for each line in the graph. For a single expression that produces multiple time series, the legend captions for those time series are in the format ***`Expression-Label Metric-Label`***. For example, if the graph includes a metric with a label of **Errors** and an expression **FILL(METRICS(), 0)** that has a label of **Filled With 0:**, one line in the legend would be **Filled With 0: Errors**. To have the legend show only the original metric labels, set *`Expression-Label`* to be empty.

   When one expression produces an array of time series on the graph, you can't change the colors used for each of those time series.

7. After you have added the desired expressions, you can simplify the graph by hiding some of the original metrics. To hide a metric or expression, clear the check box to the left of the **Id** field.

Metric math syntax and functions

The following sections explain the functions available for metric math. All functions must be written in uppercase letters (such as **AVG**), and the **Id** field for all metrics and math expressions must start with a lowercase letter.

The final result of any math expression must be a single time series or an array of time series. Some functions produce a scalar number. You can use these functions within a larger function that ultimately produces a time series. For example, taking the **AVG** of a single time series produces a scalar number, so it can't be the final expression result. But you could use it in the function **m1-AVG(m1)** to display a time series of the difference between each individual data point and the average value in the time series.

Data type abbreviations

Some functions are valid for only certain types of data. The abbreviations in the following list are used in the tables of functions to represent the types of data supported for each function:

- **S** represents a scalar number, such as 2, -5, or 50.25.
- **TS** is a time series (a series of values for a single CloudWatch metric over time): for example, the `CPUUtilization` metric for instance `i-1234567890abcdef0` over the last three days.
- **TS[]** is an array of time series, such as the time series for multiple metrics.
- **String[]** is an array of strings.

The METRICS() function

The **METRICS()** function returns all the metrics in the request. Math expressions aren't included.

You can use **METRICS()** within a larger expression that produces a single time series or an array of time series. For example, the expression **SUM(METRICS())** returns a time series (TS) that is the sum of the values of all the graphed metrics. **METRICS()/100** returns an array of time series, each of which is a time series showing each data point of one of the metrics divided by 100.

You can use the **METRICS()** function with a string to return only the graphed metrics that contain that string in their **Id** field. For example, the expression **SUM(METRICS("errors"))** returns a time series that is the sum of the values of all the graphed metrics that have ‘errors’ in their **Id** field. You can also use **SUM([METRICS(“4xx”), METRICS(“5xx”)])** to match multiple strings.

Basic arithmetic functions

The following table lists the basic arithmetic functions that are supported. Missing values in a time series are treated as 0. If the value of a data point causes a function to attempt to divide by zero, the data point is dropped.

Operation	Arguments	Examples
Arithmetic operators: + - * / ^	S, S	PERIOD(m1)/60
	S, TS	**5 * m1**
	TS, TS	**m1 - m2**

Operation	Arguments	Examples
	S, TS[] TS, TS[]	**SUM(100/[m1, m2])** **AVG(METRICS())** **METRICS()*100**
Unary subtraction -	S TS TS[]	**-5*m1** **-m1** **SUM(-[m1, m2])**

Comparison and logical operators

You can use comparison and logical operators with either a pair of time series or a pair of single scalar values. When you use a comparison operator with a pair of time series, the operators return a time series where each data point is either 0 (false) or 1 (true). If you use a comparison operator on a pair of scalar values, a single scalar value is returned, either 0 or 1.

When comparison operators are used between two time series, and only one of the time series has a value for a particular time stamp, the function treats the missing value in the other time series as **0**.

You can use logical operators in conjunction with comparison operators, to create more complex functions.

The following table lists the operators that are supported.

Type of operator	Supported operators
Comparison operators	== != <= >= <

Type of operator	Supported operators
	>
Logical operators	AND or && OR or ||

To see how these operators are used, suppose we have two time series: **metric1** has values of `[30, 20, 0, 0]` and **metric2** has values of `[20, -, 20, -]` where `-` indicates that there is no value for that timestamp.

Expression	Output
(metric1 < metric2)	**0, 0, 1, 0**
(metric1 >= 30)	**1, 0, 0, 0**
(metric1 > 15 AND metric2 > 15)	**1, 0, 0, 0**

Functions supported for metric math

The following table describes the functions that you can use in math expressions. Enter all functions in uppercase letters.

The final result of any math expression must be a single time series or an array of time series. Some functions in tables in the following sections produce a scalar number. You can use these functions within a larger function that ultimately produces a time series. For example, taking the **AVG** of a single time series produces a scalar number, so it can't be the final expression result. But you could use it in the function **m1-AVG(m1)** to display a time series of the difference between each individual data point and the average value of that data point.

In the following table, every example in the **Examples** column is an expression that results in a single time series or an array of time series. These examples show how functions that return scalar numbers can be used as part of a valid expression that produces a single time series.

Function	Arguments	Return type*	Description	Examples	Supported for cross-account?
ABS	TS TS[]	TS TS[]	Returns the absolute value of each data point.	**ABS(m1-m2)** **MIN(ABS([m1, m2]))** **ABS(METRICS())**	✓
ANOMALY_DETECTION_BAND	TS TS, S	TS[]	Returns an anomaly detection band for the specified metric. The band consists of two time series, one representing the upper limit of the "normal" expected value of the metric, and the other representing the lower limit. The function can take two arguments. The first is the ID of the metric to create the band for. The second argument is the number of standard deviations to use for the band. If you don't specify this argument, the default of 2 is used. For more information, see [Using CloudWatch anomaly detection]().	**ANOMALY_DETECTION_BAND(m1)** **ANOMALY_DETECTION_BAND(m1,4)**	

Function	Argume	Return type*	Description	Examples	Supported for cross-acc ount?
AVG	TS TS[]	S TS	The **AVG** of a single time series returns a scalar representing the average of all the data points in the metric. The **AVG** of an array of time series returns a single time series. Missing values are treated as 0. **Note** We recommend that you do not use this function in CloudWatch alarms if you want the function to return a scalar. For example, `AVG(m2)`. Whenever an alarm evaluates whether to change state, CloudWatch attempts to retrieve a higher	**SUM([m1,m2])/AVG(m2)** **AVG(METRICS())**	✓

Function	Arguments	Return type*	Description	Examples	Supported for cross-account?
			number of data points than the number specified as Evaluation Periods. This function acts differently when extra data is requested. To use this function with alarms, especially alarms that have Auto Scaling actions, we recommend that you set the alarm to use M out of N datapoints, where M < N.		
CEIL	TS TS[]	TS TS[]	Returns the ceiling of each metric. The ceiling is the smallest integer greater than or equal to each value.	**CEIL(m1)** **CEIL(METRICS())** **SUM(CEIL(METRICS()))**	✓

Function	Arguments	Return type*	Description	Examples	Supported for cross-account?
DATAPOINT_COUNT	TS TS[]	S TS	Returns a count of the data points that reported values. This is useful for calculating averages of sparse metrics. **Note** We recommend that you do not use this function in CloudWatch alarms. Whenever an alarm evaluates whether to change state, CloudWatch attempts to retrieve a higher number of data points than the number specified as Evaluation Periods. This function acts differently when	**SUM(m1) / DATAPOINT_COUNT(m1)** **DATAPOINT_COUNT(METRICS())**	✓

Function	Argume	Return type*	Description	Examples	Supported for cross-acc ount?
			extra data is requested.		

Function	Arguments	Return type*	Description	Examples	Supported for cross-account?
DB_PERF_INSIGHTS	String, String, String String, String, String[]	TS (if given a single string) TS[] (if given an array of strings)	Returns Performance Insights Counter metrics for databases such as Amazon Relational Database Service and Amazon DocumentDB (with MongoDB compatibility). This function returns the same amount of data that you can get by directly querying the Performance Insights APIs. You can use these metrics in CloudWatch for graphing and creating alarms. **Important** When you use this function, you must specify the Unique Database Resource ID of the database. This is different than	**DB_PERF_INSIGHTS('RDS', 'db-ABCDEFGHIJKLMNOPQRSTUVWXY1', 'os.cpuUtilization.user.avg')** **DB_PERF_INSIGHTS('DOCDB, 'db-ABCDEFGHIJKLMNOPQRSTUVWXY1', ['os.cpuUtilization.idle.avg', 'os.cpuUtilization.user.max'])**	

Function	Arguments	Return type*	Description	Examples	Supported for cross-account?
			the database identifier. To find the database resource ID in the Amazon RDS console, choose the DB instance to see its details. Then choose the **Configuration** tab. The **Resource ID** is displayed in the **Configuration** section. **DB_PERF_INSIGHTS** also brings in the `DBLoad` metric at sub-minute intervals. Performance Insights metrics retrieved with this function are not stored in CloudWatch. Therefore, some CloudWatch features such as cross-acc		

Function	Argume	Return type*	Description	Examples	Supported for cross-acc ount?
			ount observability, anomaly detection, metric streams, metrics explorer, and Metric Insights don't work with Performance Insights metrics that you retrieve with **DB_PERF_INSIGHTS**. A single request using the **DB_PERF_I NSIGHTS** function can retrieve the following numbers of data points. • 1080 data points for high-resolution periods (1s, 10s, 30s) • 1440 data points for standard-resolution periods (1m, 5m, 1hr, 1d) The **DB_PERF_I NSIGHTS** function supports only the following period lengths: • 1 second		

Function	Argume	Return type*	Description	Examples	Supported for cross-acc ount?
			• 10 seconds • 30 seconds • 1 minute • 5 minutes • 1 hour • 1 day For more information about Amazon RDS Performance Insights counter metrics, see [Performance Insights counter metrics](#). For more informati on about Amazon DocumentDB Performance Insights counter metrics, see [Performance Insights for counter metrics](#). **Note** High-resolution metrics with sub-minute granularity retrieved by **DB_PERF_I**		

Function	Arguments	Return type*	Description	Examples	Supported for cross-account?
			NSIGHTS are only applicable to the **DBLoad** metric, or for operating system metrics if you have enabled Enhanced Monitoring at a higher resolution. For more information about Amazon RDS enhanced monitoring, see [Monitoring OS metrics with Enhanced Monitoring](). You can create a high-resolution alarm using the **DB_PERF_INSIGHTS** function for a maximum time range of three hours. You can use the CloudWatc		

Function	Argume	Return type*	Description	Examples	Supported for cross-acc ount?
			h console to graph metrics retrieved with the **DB_PERF_I NSIGHTS** function for any time range.		
DIFF	TS TS[]	TS TS[]	Returns the difference between each value in the time series and the preceding value from that time series.	**DIFF(m1)**	✓
DIFF_TIME	TS TS[]	TS TS[]	Returns the differenc e in seconds between the timestamp of each value in the time series and the timestamp of the preceding value from that time series.	**DIFF_TIME(METRICS())**	✓

Function	Argume	Return type*	Description	Examples	Supported for cross-acc ount?
FILL	TS, [S | REPEAT | LINEAR] TS[], [TS | S | REPEAT | LINEAR]	TS TS[]	Fills the missing values of a time series. There are several options for the values to use as the filler for missing values: • You can specify a value to use as the filler value. • You can specify a metric to use as the filler value. • You can use the **REPEAT** keyword to fill missing values with the most recent actual value of the metric before the missing value. • You can use the **LINEAR** keyword to fill the missing values with values that create a linear interpolation between the values at the beginning and the end of the gap.	**FILL(m1,10)** **FILL(METRICS(), 0)** **FILL(METRICS(), m1)** **FILL(m1, MIN(m1))** **FILL(m1, REPEAT)** **FILL(METRICS(), LINEAR)**	✓

Function	Argume	Return type*	Description	Examples	Supported for cross-acc ount?
			Note When you use this function in an alarm, you can encounter an issue if your metrics are being published with a slight delay, and the most recent minute never has data. In this case, **FILL** replaces that missing data point with the requested value. That causes the latest data point for the metric to always be the FILL value, which can result in the alarm being stuck in either OK state or ALARM state. You can		

Function	Arguments	Return type*	Description	Examples	Supported for cross-account?
			work around this by using a M out of N alarm. For more information, see [Evaluating an alarm]().		
FIRST **LAST**	TS[]	TS	Returns the first or last time series from an array of time series. This is useful when used with the **SORT** function. It can also be used to get the high and low thresholds from the **ANOMALY_DETECTION_BAND** function.	**IF(FIRST(SORT(METRICS(), AVG, DESC))>100, 1, 0)** Looks at the top metric from an array, which is sorted by AVG. It then returns a 1 or 0 for each data point, depending on whether that data point value is more than 100. **LAST(ANOMALY_DETECTION_BAND(m1))** returns the upper bound of the anomaly prediction band.	✓
FLOOR	TS TS[]	TS TS[]	Returns the floor of each metric. The floor is the largest integer less than or equal to each value.	**FLOOR(m1)** **FLOOR(METRICS())**	✓

Function	Arguments	Return type*	Description	Examples	Supported for cross-account?
IF	**IF** expression	TS	Use **IF** along with a comparison operator to filter out data points from a time series, or create a mixed time-series composed of multiple collated time series. For more information, see Using IF expressions.	For examples, see Using IF expressions.	✓
INSIGHT_RULE_METRIC	**INSIGHT_RULE_METRIC(ruleName, metricName)**	TS	Use **INSIGHT_RULE_METRIC** to extract statistics from a rule in Contributor Insights. For more information, see Graphing metrics generated by rules in CloudWatch.		
LAMBDA	**LAMBDA(LambdaFunctionName [, optional-arg]*)**	TS TS[]	Calls a Lambda function to query metrics from a data source that is not CloudWatch. For more information, see How to pass arguments to your Lambda function.		

Function	Arguments	Return type*	Description	Examples	Supported for cross-account?
LOG	TS TS[]	TS TS[]	The **LOG** of a time series returns the natural logarithm value of each value in the time series.	**LOG(METRICS())**	✓
LOG10	TS TS[]	TS TS[]	The **LOG10** of a time series returns the base-10 logarithm value of each value in the time series.	**LOG10(m1)**	✓

Function	Argume	Return type*	Description	Examples	Supported for cross-acc ount?
MAX	TS TS[]	S TS	The **MAX** of a single time series returns a scalar representing the maximum value of all data points in the metric. If you input an array of time series, the **MAX** function creates and returns a time series that consists of the highest value for each data point, among the time series that were used as the input. **Note** We recommend that you do not use this function in CloudWatc h alarms if you want the function to return a scalar. For example, `MAX(m2)` Whenever an	**MAX(m1)/m1** **MAX(METRICS())**	✓

Function	Argume	Return type*	Description	Examples	Supported for cross-acc ount?
			alarm evaluates whether to change state, CloudWatch attempts to retrieve a higher number of data points than the number specified as Evaluation Periods. This function acts differently when extra data is requested.		
METRIC_CO UNT	TS[]	S	Returns the number of metrics in the time series array.	**m1/METRIC _COUNT(METRICS())**	✓

Function	Argume	Return type*	Description	Examples	Supported for cross-acc ount?
METRICS	null string	TS[]	The **METRICS()** function returns all CloudWatch metrics in the request. Math expressions aren't included. You can use **METRICS()** within a larger expression that produces a single time series or an array of time series. You can use the **METRICS()** function with a string to return only the graphed metrics that contain that string in their **Id** field. For example, the expression **SUM(METRICS("errors"))** returns a time series that is the sum of the values of all the graphed metrics that have 'errors' in their **Id** field. You can also use **SUM([METRICS("4xx"), METRICS("5xx")])** to match multiple strings.	**AVG(METRICS())** **SUM(METRICS("errors"))**	✓

Function	Argume	Return type*	Description	Examples	Supported for cross-acc ount?
MIN	TS TS[]	S TS	The **MIN** of a single time series returns a scalar representing the minimum value of all data points in the metric. If you input an array of time series, the **MIN** function creates and returns a time series that consists of the lowest value for each data point, among the time series that were used as the input. If you input an array of time series, the **MIN** function creates and returns a time series that consists of the lowest value for each data point, among the time series that were used as the input. **Note** We recommend that you do	**m1-MIN(m1)** **MIN(METRICS())**	✓

Function	Arguments	Return type*	Description	Examples	Supported for cross-account?
			not use this function in CloudWatch alarms if you want the function to return a scalar. For example, `MIN(m2)` Whenever an alarm evaluates whether to change state, CloudWatch attempts to retrieve a higher number of data points than the number specified as Evaluation Periods. This function acts differently when extra data is requested.		

Function	Argume	Return type*	Description	Examples	Supported for cross-acc ount?
MINUTE **HOUR** **DAY** **DATE** **MONTH** **YEAR** **EPOCH**	TS	TS	These functions take the period and range of the time series and return a new non-spars e time series where each value is based on its timestamp. • **MINUTE** returns a non-sparse time series of integers between 0 and 59 that represent the UTC minute of each timestamp in the original time series. • **HOUR** returns a non-sparse time series of integers between 0 and 23 that represent the UTC hour of each timestamp in the original time series. • **DAY** returns a non-sparse time series of integers between 1 and 7 that represent the UTC day of the week of each timestamp in the	**MINUTE(m1)** **IF(DAY(m1)<6,m1)** returns metrics only from weekdays, Monday to Friday. **IF(MONTH(m1) == 4,m1)** returns only metrics published in April.	✓

Function	Argume	Return type*	Description	Examples	Supported for cross-acc ount?
			original time series. 1 represents Monday and 7 represents Sunday. • **DATE** returns a non-sparse time series of integers between 1 and 31 that represent the UTC day of the month of each timestamp in the original time series. • **MONTH** returns a non-sparse time series of integers between 1 and 12 that represent the UTC month of each timestamp in the original time series. 1 represents January and 12 represents December. • **YEAR** returns a non-sparse time series of integers that represent the UTC year of each timestamp in the original time series.		

Function	Arguments	Return type*	Description	Examples	Supported for cross-account?
			• **EPOCH** returns a non-sparse time series of integers that represent the UTC time in seconds since the Epoch of each timestamp in the original time series. The Epoch is January 1, 1970.		
PERIOD	TS	S	Returns the period of the metric in seconds. Valid input is metrics, not the results of other expressions.	**m1/PERIOD(m1)**	✓

Function	Argume	Return type*	Description	Examples	Supported for cross-acc ount?
RATE	TS TS[]	TS TS[]	Returns the rate of change of the metric per second. This is calculated as the difference between the latest data point value and the previous data point value, divided by the time difference in seconds between the two values. **Important** Setting alarms on expressio ns that use the RATE function on metrics with sparse data can behave unpredictably, because the range of data points fetched when evaluatin g the alarm can vary based on when the data	**RATE(m1)** **RATE(METRICS())**	✓

Function	Argume	Return type*	Description	Examples	Supported for cross-acc ount?
			points were last published.		

Function	Argume	Return type*	Description	Examples	Supported for cross-acc ount?
REMOVE_EMPTY	TS[]	TS[]	Removes any time series that have no data points from an array of time series. The result is an array of time series where each time series contains at least one data point. **Note** We recommend that you do not use this function in CloudWatch alarms. Whenever an alarm evaluates whether to change state, CloudWatch attempts to retrieve a higher number of data points than the number specified as Evaluation Periods. This function acts	**REMOVE_EMPTY(METRICS())**	✓

Function	Argume	Return type*	Description	Examples	Supported for cross-acc ount?
			differently when extra data is requested.		

Function	Argume	Return type*	Description	Examples	Supported for cross-acc ount?
RUNNING_S UM	TS TS[]	TS TS[]	Returns a time series with the running sum of the values in the original time series. **Note** We recommend that you do not use this function in CloudWatc h alarms. Whenever an alarm evaluates whether to change state, CloudWatch attempts to retrieve a higher number of data points than the number specified as Evaluation Periods. This function acts differently when extra data is requested.	**RUNNING_S UM([m1,m2])**	✓

Function	Argument	Return type*	Description	Examples	Supported for cross-account?
SEARCH	Search expression	One or more TS	Returns one or more time series that match a search criteria that you specify. The **SEARCH** function enables you to add multiple related time series to a graph with one expression. The graph is dynamically updated to include new metrics that are added later and match the search criteria. For more information, see [Use search expressions in graphs](). You can't create an alarm based on a **SEARCH** expression. This is because search expressions return multiple time series, and an alarm based on a math expression can watch only one time series. If you are signed in to a monitoring account in CloudWatch cross-		✓

Function	Arguments	Return type*	Description	Examples	Supported for cross-account?
			account observability, the **SEARCH** function finds metrics in the source accounts and the monitoring account.		
SERVICE_QUOTA	TS that is a usage metric	TS	Returns the service quota for the given usage metric. You can use this to visualize how your current usage compares to the quota, and to set alarms that warn you when you approach the quota. For more information, see [Amazon usage metrics]().		✓

Function	Arguments	Return type*	Description	Examples	Supported for cross-account?
SLICE	(TS[], S, S) or (TS[], S)	TS[] TS	Retrieves part of an array of time series. This is especially useful when combined with **SORT**. For example, you can exclude the top result from an array of time series. You can use two scalar arguments to define the set of time series that you want returned. The two scalars define the start (inclusive) and end (exclusive) of the array to return. The array is zero-indexed, so the first time series in the array is time series 0. Alternatively, you can specify just one value, and CloudWatch returns all time series starting with that value.	**SLICE(SORT(METRICS (), SUM, DESC), 0, 10)** returns the 10 metrics from the array of metrics in the request that have the highest SUM value. **SLICE(SORT(METRICS (), AVG, ASC), 5)** sorts the array of metrics by the AVG statistic, then returns all the time series except for the 5 with the lowest AVG.	✓

Function	Argume	Return type*	Description	Examples	Supported for cross-acc ount?
SORT	(TS[], FUNCTI SORT_C R) (TS[], FUNCTI SORT_C R, S)	TS[]	Sorts an array of time series according to the function you specify. The function you use can be **AVG**, **MIN**, **MAX**, or **SUM**. The sort order can be either **ASC** for ascending (lowest values first) or **DESC** to sort the higher values first. You can optionall y specify a number after the sort order which acts as a limit. For example, specifying a limit of 5 returns only the top 5 time series from the sort. When this math function is displayed on a graph, the labels for each metric in the graph are also sorted and numbered.	**SORT(METRICS(), AVG, DESC, 10)** calculates the average value of each time series, sorts the time series with the highest values at the beginning of the sort, and returns only the 10 time series with the highest averages. **SORT(METRICS(), MAX, ASC)** sorts the array of metrics by the MAX statistic, then returns all of them in ascending order.	✓

Function	Argume	Return type*	Description	Examples	Supported for cross-acc ount?
STDDEV	TS TS[]	S TS	The **STDDEV** of a single time series returns a scalar representing the standard deviation of all data points in the metric. The **STDDEV** of an array of time series returns a single time series. **Note** We recommend that you do not use this function in CloudWatc h alarms if you want the function to return a scalar. For example, `STDDEV(m2)` Whenever an alarm evaluates whether to change state, CloudWatch attempts to retrieve a higher number of data	**m1/STDDEV(m1)** **STDDEV(METRICS())**	✓

Function	Argume	Return type*	Description	Examples	Supported for cross-acc ount?
			points than the number specified as Evaluation Periods. This function acts differently when extra data is requested.		

Function	Argume	Return type*	Description	Examples	Supported for cross-acc ount?
SUM	TS TS[]	S TS	The **SUM** of a single time series returns a scalar representing the sum of the values of all data points in the metric. The **SUM** of an array of time series returns a single time series. **Note** We recommend that you do not use this function in CloudWatc h alarms if you want the function to return a scalar. For example, `SUM(m1)`. Whenever an alarm evaluates whether to change state, CloudWatch attempts to retrieve a higher number of data	**SUM(METRICS())/SUM (m1)** **SUM([m1,m2])** **SUM(METRICS("error s"))/SUM(METRICS(" requests"))*100**	✓

Function	Arguments	Return type*	Description	Examples	Supported for cross-account?
			points than the number specified as Evaluation Periods. This function acts differently when extra data is requested.		
TIME_SERIES	S	TS	Returns a non-sparse time series where every value is set to a scalar argument.	**TIME_SERIES(MAX(m1))** **TIME_SERIES(5*AVG(m1))** **TIME_SERIES(10)**	✓

*Using a function that returns only a scalar number is not valid, as all final results of expressions must be a single time series or an array of time series. Instead, use these functions as part of a larger expression that returns a time series.

Using IF expressions

Use **IF** along with a comparison operator to filter out data points from a time series, or create a mixed time-series composed of multiple collated time series.

IF uses the following arguments:

```
IF(condition, trueValue, falseValue)
```

The condition evaluates to FALSE if the value of the condition data point is 0, and to TRUE if the value of the condition is any other value, whether that value is positive or negative. If the condition is a time series, it is evaluated separately for every timestamp.

The following lists the valid syntax's. For each of these syntax's, the output is a single time series.

- **IF(TS *`Comparison Operator`* S, S | TS, *S* / *TS*)**

> **Note**
>
> If the `TS comparison operator S` is TRUE but `metric2` doesn't have a corresponding data point, the output will be 0.

- **IF(TS, TS, *TS*)**
- **IF(TS, S, *TS*)**
- **IF(TS, TS, S)**
- **IF(TS, S, S)**
- **IF(S, TS, *TS*)**

The following sections provide more details and examples for these syntax's.

IF(TS *`Comparison Operator`* S, scalar2 | metric2, scalar3 | metric3)

The corresponding output time series value:

- has the value of **scalar2** or **metric2**, if **TS** *`Comparison Operator`* **S** is TRUE
- has the value of **scalar3** or **metric3**, if **TS** *`Comparison Operator`* **S** is FALSE
- has the value of **0** if the **TS** *`Comparison Operator`* is TRUE and the corresponding data point in **metric2** doesn't exist.
- has the value of **0** if the **TS** *`Comparison Operator`* is FALSE and the corresponding data point in **metric3** doesn't exist.
- is an empty time series, if the corresponding data point of does not exist in **metric3**, or if **scalar3/metric3** is omitted from the expression

IF(metric1, metric2, *metric3*)

For each data point of **metric1**, the corresponding output time series value:

- has the value of **metric2**, if the corresponding data point of **metric1** is TRUE.
- has the value of **metric3**, if the corresponding data point of **metric1** is FALSE.
- has the value of **0**, if the corresponding data point of **metric1** is TRUE and the corresponding data point does not exist in **metric2**.
- is dropped, if the corresponding data point of **metric1** is FALSE and the corresponding data point does not exist in **metric3**
- is dropped, if the corresponding data point of **metric1** is FALSE and **metric3** is omitted from the expression.
- is dropped, if the corresponding data point of **metric1** is missing.

The following table shows an example for this syntax.

Metric or function	Values
(metric1)	**[1, 1, 0, 0, -]**
(metric2)	**[30, -, 0, 0, 30]**
(metric3)	**[0, 0, 20, -, 20]**
IF(metric1, metric2, metric3)	**[30, 0, 20, 0, -]**

IF(metric1, scalar2, metric3)

For each data point of **metric1**, the corresponding output time series value:

- has the value of **scalar2**, if the corresponding data point of **metric1** is TRUE.
- has the value of **metric3**, if the corresponding data point of **metric1** is FALSE.
- is dropped, if the corresponding data point of **metric1** is FALSE and the corresponding data point does not exist on **metric3**, or if **metric3** is omitted from the expression.

Metric or function	Values
(metric1)	**[1, 1, 0, 0, -]**

Metric or function	Values
scalar2	**5**
(metric3)	**[0, 0, 20, -, 20]**
IF(metric1, scalar2, metric3)	**[5, 5, 20, -, -]**

IF(metric1, metric2, scalar3)

For each data point of **metric1**, the corresponding output time series value:

- has the value of **metric2**, if the corresponding data point of **metric1** is TRUE.
- has the value of **scalar3**, if the corresponding data point of **metric1** is FALSE.
- has the value of **0**, if the corresponding data point of **metric1** is TRUE and the corresponding data point does not exist in **metric2**.
- is dropped if the corresponding data point in **metric1** does not exist.

Metric or function	Values
(metric1)	**[1, 1, 0, 0, -]**
(metric2)	**[30, -, 0, 0, 30]**
scalar3	**5**
IF(metric1, metric2, scalar3)	**[30, 0, 5, 5, -]**

IF(scalar1, metric2, *metric3*)

The corresponding output time series value:

- has the value of **metric2**, if **scalar1** is TRUE.
- has the value of **metric3**, if **scalar1** is FALSE.
- is an empty time series, if **metric3** is omitted from the expression.

Use case examples for IF expressions

The following examples illustrate the possible uses of the **IF** function.

- To display only the low values of a metric:

  IF(metric1<400, metric1)

- To change each data point in a metric to one of two values, to show relative highs and lows of the original metric:

  IF(metric1<400, 10, 2)

- To display a 1 for each timestamp where latency is over the threshold, and display a 0 for all other data points:

  IF(latency>threshold, 1, 0)

Use metric math with the GetMetricData API operation

You can use `GetMetricData` to perform calculations using math expressions, and also retrieve large batches of metric data in one API call. For more information, see GetMetricData.

Anomaly detection on metric math

Anomaly detection on metric math is a feature that you can use to create anomaly detection alarms on single metrics and the outputs of metric math expressions. You can use these expressions to create graphs that visualize anomaly detection bands. The feature supports basic arithmetic functions, comparison and logical operators, and most other functions.

Anomaly detection on metric math doesn't support the following functions:

- Expressions that contain more than one **ANOMALY_DETECTION_BAND** in the same line.
- Expressions that contain more than 10 metrics or math expressions.
- Expressions that contain the **METRICS** expression.
- Expressions that contain the **SEARCH** function.
- Expressions that use the **DP_PERF_INSIGHTS** function.
- Expressions that use metrics with different periods.
- Expressions that use periods longer than one hour as input.

- Expressions that use high-resolution metrics as input.

For more information about this feature, see Using CloudWatch anomaly detection in the *Amazon CloudWatch User Guide*.

Use search expressions in graphs

Search expressions are a type of math expression that you can add to CloudWatch graphs. Search expressions enable you to quickly add multiple related metrics to a graph. They also enable you to create dynamic graphs that automatically add appropriate metrics to their display, even if those metrics don't exist when you first create the graph.

For example, you can create a search expression that displays the `AWS/EC2 CPUUtilization` metric for all instances in the Region. If you later launch a new instance, the `CPUUtilization` of the new instance is automatically added to the graph.

When you use a search expression in a graph, the search finds the search expression in metric names, namespaces, dimension names, and dimension values. You can use Boolean operators for more complex and powerful searches. A search expression can find only metrics that have reported data within the past two weeks.

You can't create an alarm based on the **SEARCH** expression. This is because search expressions return multiple time series, and an alarm based on a math expression can watch only one time series.

If you are using a monitoring account in CloudWatch cross-account observability, your search expressions can find metrics in the source accounts linked to that monitoring account.

Topics

CloudWatch search expression syntax

A valid search expression has the following format.

```
SEARCH(' {Namespace, DimensionName1, DimensionName2, ...} SearchTerm', 'Statistic')
```

For example:

```
SEARCH('{AWS/EC2,InstanceId} MetricName="CPUUtilization"', 'Average')
```

- The first part of the query after the word `SEARCH`, enclosed in curly braces, is the *metric schema* to be searched. The metric schema contains a metric namespace and one or more dimension names. Including a metric schema in a search query is optional. If specified, the metric schema must contain a namespace and can optionally contain one or more dimension names that are valid in that namespace.

  You don't need to use quote marks inside the metric schema unless a namespace or dimension name includes spaces or non-alphanumeric characters. In that case, you must enclose the name that contains those characters with double quotes.
- The `SearchTerm` is also optional, but a valid search must contain either the metric schema, the `SearchTerm`, or both. The `SearchTerm` usually contains one or more account IDs, metric names or dimension values. The `SearchTerm` can include multiple terms to search for, by both partial match and exact match. It can also contain Boolean operators.

  Using an account ID in a `SearchTerm` works only in accounts that are set up as monitoring accounts for CloudWatch cross-account observability. The syntax for an account ID in `SearchTerm` is `:aws.AccountId = `*444455556666*. You can also use `'LOCAL'` to specify the monitoring account itself: `:aws.AccountId = 'LOCAL'`

  For more information, see CloudWatch cross-account observability.

  The `SearchTerm` can include one or more designators, such as `MetricName=` as in this example, but using designators isn't required.

  The metric schema and `SearchTerm` must be enclosed together in a pair of single quote marks.
- The `Statistic` is the name of any valid CloudWatch statistic. It must be enclosed by single quotes. For more information, see Statistics.

The preceding example searches the `AWS/EC2` namespace for any metrics that have `InstanceId` as a dimension name. It returns all `CPUUtilization` metrics that it finds, with the graph showing the `Average` statistic.

A search expression can find only metrics that have reported data within the past two weeks.

Search expression limits

The maximum search expression query size is 1024 characters. You can have as many as 100 search expressions on one graph. A graph can display as many as 500 time series.

CloudWatch search expressions: Tokenization

When you specify a `SearchTerm`, the search function searches for *tokens*, which are substrings that CloudWatch automatically generates from full metric names, dimension names, dimension values, and namespaces. CloudWatch generates tokens distinguished by the camel-case capitalization in the original string. Numeric characters also serve as the start of new tokens, and non-alphanumeric characters serve as delimiters, creating tokens before and after the non-alphanumeric characters.

A continuous string of the same type of token delimiter character results in one token.

All generated tokens are in lowercase. The following table shows some examples of tokens generated.

Original string	Tokens generated
CustomCount1	`customcount1` , `custom`, `count`, `1`
SDBFailure	`sdbfailure` , `sdb`, `failure`
Project2-trial333	`project2trial333` , `project`, `2`, `trial`, `333`

CloudWatch search expressions: Partial matches

When you specify a `SearchTerm`, the search term is also tokenized. CloudWatch finds metrics based on partial matches, which are matches of a single token generated from the search term to a single token generated from a metric name, namespace, dimension name, or dimension value.

Partial match searches to match a single token are case insensitive. For example, using any of the following search terms can return the `CustomCount1` metric:

- `count`

- `Count`
- `COUNT`

However, using `couNT` as a search term doesn't find `CustomCount1` because the capitalization in the search term `couNT` is tokenized into `cou` and `NT`.

Searches can also match composite tokens, which are multiple tokens that appear consecutively in the original name. To match a composite token, the search is case sensitive. For example, if the original term is `CustomCount1`, searches for `CustomCount` or `Count1` are successful, but searches for `customcount` or `count1` aren't.

CloudWatch search expressions: Exact matches

You can define a search to find only exact matches of your search term by using double quotes around the part of the search term that requires an exact match. These double-quotes are enclosed in the single-quotes used around the entire search term. For example, **`SEARCH(' {MyNamespace}, "CustomCount1" ', 'Maximum')`** finds the exact string `CustomCount1` if it exists as a metric name, dimension name, or dimension value in the namespace named `MyNamespace`. However, the searches **`SEARCH(' {MyNamespace}, "customcount1" ', 'Maximum')`** or **`SEARCH(' {MyNamespace}, "Custom" ', 'Maximum')`** do not find this string.

You can combine partial match terms and exact match terms in a single search expression. For example, **`SEARCH(' {AWS/NetworkELB, LoadBalancer} "ConsumedLCUs" OR flow ', 'Maximum')`** returns the Elastic Load Balancing metric named `ConsumedLCUs` as well as all Elastic Load Balancing metrics or dimensions that contain the token `flow`.

Using exact match is also a good way to find names with special characters, such as non-alphanumeric characters or spaces, as in the following example.

```
SEARCH(' {"My Namespace", "Dimension@Name"}, "Custom:Name[Special_Characters" ',
 'Maximum')
```

CloudWatch search expressions: Excluding a metric schema

All examples shown so far include a metric schema, in curly braces. Searches that omit a metric schema are also valid.

For example, **SEARCH(' "CPUUtilization" ', 'Average')** returns all metric names, dimension names, dimension values, and namespaces that are an exact match for the string `CPUUtilization`. In the Amazon metric namespaces, this can include metrics from several services including Amazon EC2, Amazon ECS, SageMaker AI, and others.

To narrow this search to only one Amazon service, the best practice is to specify the namespace and any necessary dimensions in the metric schema, as in the following example. Although this narrows the search to the `AWS/EC2` namespace, it would still return results of other metrics if you have defined `CPUUtilization` as a dimension value for those metrics.

```
SEARCH(' {AWS/EC2, InstanceType} "CPUUtilization" ', 'Average')
```

Alternatively you could add the namespace in the `SearchTerm` as in the following example. But in this example, the search would match any `AWS/EC2` string, even if it was a custom dimension name or value.

```
SEARCH(' "AWS/EC2" MetricName="CPUUtilization" ', 'Average')
```

CloudWatch search expressions: Specifying property names in the search

The following exact match search for `"CustomCount1"` returns all metrics with exactly that name.

```
SEARCH(' "CustomCount1" ', 'Maximum')
```

But it also returns metrics with dimension names, dimension values, or namespaces of `CustomCount1`. To structure your search further, you can specify the property name of the type of object that you want to find in your searches. The following example searches all namespaces and returns metrics named `CustomCount1`.

```
SEARCH(' MetricName="CustomCount1" ', 'Maximum')
```

You can also use namespaces and dimension name/value pairs as property names, as in the following examples. The first of these examples also illustrates that you can use property names with partial match searches as well.

```
SEARCH(' InstanceType=micro ', 'Average')
```

```
SEARCH(' InstanceType="t2.micro" Namespace="AWS/EC2" ', 'Average')
```

CloudWatch search expressions: Non-alphanumeric characters

Non-alphanumeric characters serve as delimiters, and mark where the names of metrics, dimensions, namespaces, and search terms are to be separated into tokens. When terms are tokenized, non-alphanumeric characters are stripped out and don't appear in the tokens. For example, `Network-Errors_2` generates the tokens `network`, `errors`, and `2`.

Your search term can include any non-alphanumeric characters. If these characters appear in your search term, they can specify composite tokens in a partial match. For example, all of the following searches would find metrics named either `Network-Errors-2` or `NetworkErrors2`.

```
network/errors
network+errors
network-errors
Network_Errors
```

When you're doing an exact value search, any non-alphanumeric characters used in the exact search must be the correct characters that appear in the string being searched for. For example, if you want to find `Network-Errors-2`, searching for `"Network-Errors-2"` is successful, but a search for `"Network_Errors_2"` isn't.

When you perform an exact match search, the following characters must be escaped with a backslash.

```
" \ ( )
```

For example, to find the metric name `Europe\France Traffic(Network)` by exact match, use the search term **`"Europe\\France Traffic\(Network\)"`**

CloudWatch search expressions: Boolean operators

Search supports the use of the Boolean operators `AND`, `OR`, and `NOT` within the `SearchTerm`. Boolean operators are enclosed in the single quote marks that you use to enclose the entire search term. Boolean operators are case sensitive, so `and`, `or`, and `not` aren't valid as Boolean operators.

You can use `AND` explicitly in your search, such as **`SEARCH('{AWS/EC2,InstanceId} network AND packets', 'Average')`**. Not using any Boolean operator between search terms implicitly searches them as if there were an `AND` operator, so **`SEARCH(' {AWS/EC2,InstanceId} network packets ', 'Average')`** yields the same search results.

Use NOT to exclude subsets of data from the results. For example, **SEARCH(' {AWS/EC2,InstanceId} MetricName="CPUUtilization" NOT i-1234567890123456 ', 'Average')** returns the CPUUtilization for all your instances, except for the instance i-1234567890123456. You can also use a NOT clause as the only search term. For example, **SEARCH('NOT Namespace=AWS ', 'Maximum')** yields all your custom metrics (metrics with namespaces that don't include AWS).

You can use multiple NOT phrases in a query. For example, **SEARCH(' {AWS/EC2,InstanceId} MetricName="CPUUtilization" NOT "ProjectA" NOT "ProjectB" ', 'Average')** returns the CPUUtilization of all instances in the Region, except for those with dimension values of ProjectA or ProjectB.

You can combine Boolean operators for more powerful and detailed searches, as in the following examples. Use parentheses to group the operators.

Both of the next two examples return all metric names containing ReadOps from both the EC2 and EBS namespaces.

```
SEARCH(' (EC2 OR EBS) AND MetricName=ReadOps ', 'Maximum')
```

```
SEARCH(' (EC2 OR EBS) MetricName=ReadOps ', 'Maximum')
```

The following example narrows the previous search to only results that include ProjectA, which could be the value of a dimension.

```
SEARCH(' (EC2 OR EBS) AND ReadOps AND ProjectA ', 'Maximum')
```

The following example uses nested grouping. It returns Lambda metrics for Errors from all functions, and Invocations of functions with names that include the strings ProjectA or ProjectB.

```
SEARCH(' {AWS/Lambda,FunctionName} MetricName="Errors" OR (MetricName="Invocations" AND
 (ProjectA OR ProjectB)) ', 'Average')
```

CloudWatch search expressions: Using math expressions

You can use a search expression within a math expressions in a graph.

For example, **`SUM(SEARCH(' {AWS/Lambda, FunctionName} MetricName="Errors" ', 'Sum'))`** returns the sum of the `Errors` metric of all your Lambda functions.

Using separate lines for your search expression and math expression might yield more useful results. For example, suppose that you use the following two expressions in a graph. The first line displays separate `Errors` lines for each of your Lambda functions. The ID of this expression is `e1`. The second line adds another line showing the sum of the errors from all of the functions.

```
SEARCH(' {AWS/Lambda, FunctionName}, MetricName="Errors" ', 'Sum')
SUM(e1)
```

CloudWatch search expression examples

The following examples illustrate more search expression uses and syntax. Let's start with a search for `CPUUtilization` across all instances in the Region and then look at variations.

This example displays one line for each instance in the Region, showing the `CPUUtilization` metric from the `AWS/EC2` namespace.

```
SEARCH(' {AWS/EC2,InstanceId} MetricName="CPUUtilization" ', 'Average')
```

Changing `InstanceId` to `InstanceType` changes the graph to show one line for each instance type used in the Region. Data from all instances of each type is aggregated into one line for that instance type.

```
SEARCH(' {AWS/EC2,InstanceType} MetricName="CPUUtilization" ', 'Average')
```

The following example aggregates the `CPUUtilization` by instance type and displays one line for each instance type that includes the string `micro`.

```
SEARCH('{AWS/EC2,InstanceType} InstanceType=micro MetricName="CPUUtilization" ',
 'Average')
```

This example narrows the previous example, changing the `InstanceType` to an exact search for t2.micro instances.

```
SEARCH('{AWS/EC2,InstanceType} InstanceType="t2.micro" MetricName="CPUUtilization" ',
 'Average')
```

The following search removes the `{metric schema}` part of the query, so the `CPUUtilization` metric from all namespaces appears in the graph. This can return quite a few results because the graph includes multiple lines for the `CPUUtilization` metric from each Amazon service, aggregated along different dimensions.

```
SEARCH('MetricName="CPUUtilization" ', 'Average')
```

To narrow these results a bit, you can specify two specific metric namespaces.

```
SEARCH('MetricName="CPUUtilization" AND ("AWS/ECS" OR "AWS/ES") ', 'Average')
```

The preceding example is the only way to do a search of specific multiple namespaces with one search query, as you can specify only one metric schema in each query. However, to add more structure, you could use two queries in the graph, as in the following example. This example also adds more structure by specifying a dimension to use to aggregate the data for Amazon ECS.

```
SEARCH('{AWS/ECS ClusterName}, MetricName="CPUUtilization" ', 'Average')
SEARCH(' {AWS/EBS} MetricName="CPUUtilization" ', 'Average')
```

The following example returns the Elastic Load Balancing metric named `ConsumedLCUs` as well as all Elastic Load Balancing metrics or dimensions that contain the token `flow`.

```
SEARCH('{AWS/NetworkELB, LoadBalancer} "ConsumedLCUs" OR flow ', 'Maximum')
```

The following example uses nested grouping. It returns Lambda metrics for `Errors` from all functions and `Invocations` of functions with names that include the strings `ProjectA` or `ProjectB`.

```
SEARCH('{AWS/Lambda,FunctionName} MetricName="Errors" OR (MetricName="Invocations" AND
 (ProjectA OR ProjectB)) ', 'Average')
```

The following example displays all of your custom metrics, excluding metrics generated by Amazon services.

```
SEARCH('NOT Namespace=AWS ', 'Average')
```

The following example displays metrics with metric names, namespaces, dimension names, and dimension values that contain the string `Errors` as part of their name.

```
SEARCH('Errors', 'Average')
```

The following example narrows that search to exact matches. For example, this search finds the metric name `Errors` but not metrics named `ConnectionErrors` or `errors`.

```
SEARCH(' "Errors" ', 'Average')
```

The following example shows how to specify names that contain spaces or special characters in the metric schema part of the search term.

```
SEARCH('{"Custom-Namespace", "Dimension Name With Spaces"}, ErrorCount ', 'Maximum')
```

CloudWatch cross-account observability search expression examples

CloudWatch cross-account observability examples

If you are signed in to an account that is set up as a monitoring account in CloudWatch cross-account observability, you can use the **SEARCH** function to return metrics from specified source accounts. For more information, see CloudWatch cross-account observability.

The following example retrieves all Lambda metrics from the account with the account ID 111122223333.

```
SEARCH(' AWS/Lambda :aws.AccountId = 111122223333 ', 'Average')
```

The following example retrieves all `AWS/EC2` metrics from two accounts: 111122223333 and 777788889999.

```
SEARCH(' AWS/EC2 :aws.AccountId = (111122223333 OR 777788889999) ', 'Average')
```

The following example retrieves all `AWS/EC2` metrics from the source account 111122223333 and from the monitoring account itself.

```
SEARCH(' AWS/EC2 :aws.AccountId = (111122223333 OR 'LOCAL') ', 'Average')
```

The following example retrieves the `SUM` of the `MetaDataToken` metric from the account `444455556666` with the `InstanceId` dimension.

```
SEARCH('{AWS/EC2,InstanceId} :aws.AccountId=444455556666 MetricName=\"MetadataNoToken
\"','Sum')
```

Create a CloudWatch graph with a search expression

On the CloudWatch console, you can access search capability when you add a graph to a dashboard, or by using the **Metrics** view.

You can't create an alarm based on a **SEARCH** expression. This is because search expressions return multiple time series, and an alarm based on a math expression can watch only one time series.

To add a graph with a search expression to an existing dashboard

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Dashboards** and select a dashboard.
3. Choose **Add widget**.
4. Choose either **Line** or **Stacked area** and choose **Configure**.
5. On the **Graphed metrics** tab, choose **Add a math expression**.
6. For **Details**, enter the search expression that you want. For example, **`SEARCH('{AWS/EC2,InstanceId} MetricName="CPUUtilization"', 'Average')`**
7. (Optional) To add another search expression or math expression to the graph, choose **Add a math expression**
8. (Optional) After you add a search expression, you can specify a dynamic label to appear on the graph legend for each metric. Dynamic labels display a statistic about the metric and automatically update when the dashboard or graph is refreshed. To add a dynamic label, choose **Graphed metrics** and then **Dynamic labels**.

   By default, the dynamic values you add to the label appear at the beginning of the label. You can then click the **Label** value for the metric to edit the label. For more information, see Use dynamic labels.
9. (Optional) To add a single metric to the graph, choose the **All metrics** tab and drill down to the metric you want.
10. (Optional) To change the time range shown on the graph, choose either **custom** at the top of the graph or one of the time periods to the left of **custom**.

11. (Optional) Horizontal annotations help dashboard users quickly see when a metric has spiked to a certain level or whether the metric is within a predefined range. To add a horizontal annotation, choose **Graph options** and then **Add horizontal annotation**:

    a. For **Label**, enter a label for the annotation.

    b. For **Value**, enter the metric value where the horizontal annotation appears.

    c. For **Fill**, specify whether to use fill shading with this annotation. For example, choose `Above` or `Below` for the corresponding area to be filled. If you specify `Between`, another `Value` field appears, and the area of the graph between the two values is filled.

    d. For **Axis**, specify whether the numbers in `Value` refer to the metric associated with the left y-axis or the right y-axis if the graph includes multiple metrics.

       You can change the fill color of an annotation by choosing the color square in the left column of the annotation.

    Repeat these steps to add multiple horizontal annotations to the same graph.

    To hide an annotation, clear the check box in the left column for that annotation.

    To delete an annotation, choose **x** in the **Actions** column.

12. (Optional) Vertical annotations help you mark milestones in a graph, such as operational events or the beginning and end of a deployment. To add a vertical annotation, choose **Graph options** and then **Add vertical annotation**:

    a. For **Label**, enter a label for the annotation. To show only the date and time on the annotation, keep the **Label** field blank.

    b. For **Date**, specify the date and time where the vertical annotation appears.

    c. For **Fill**, specify whether to use fill shading before or after a vertical annotation or between two vertical annotations. For example, choose `Before` or `After` for the corresponding area to be filled. If you specify `Between`, another `Date` field appears, and the area of the graph between the two values is filled.

    Repeat these steps to add multiple vertical annotations to the same graph.

    To hide an annotation, clear the check box in the left column for that annotation.

    To delete an annotation, choose **x** in the **Actions** column.

13. Choose **Create widget**.
14. Choose **Save dashboard**.

To use the Metrics view to graph searched metrics

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Metrics**, **All metrics**.
3. In the search field, enter the tokens to search for: for example, **`cpuutilization t2.small`**.

   Results that match your search appear.
4. To graph all of the metrics that match your search, choose **Graph search**.

   or

   To refine your search, choose one of the namespaces that appeared in your search results.
5. If you selected a namespace to narrow your results, you can do the following:

   a. To graph one or more metrics, select the check box next to each metric. To select all metrics, select the check box in the heading row of the table.
   b. To refine your search, hover over a metric name and choose **Add to search** or **Search for this only**.
   c. To view help for a metric, select the metric name and choose **What is this?**.

   The selected metrics appear on the graph.
6. (Optional) Select one of the buttons in the search bar to edit that part of the search term.
7. (Optional) To add the graph to a dashboard, choose **Actions** and then **Add to dashboard**.

Get statistics for a metric

CloudWatch statistics definitions

Statistics are metric data aggregations over specified periods of time. When you graph or retrieve the statistics for a metric, you specify the *Period* of time, such as five minutes, to use to calculate each statistical value. For example, if the **Period** is five minutes, the **Sum** is the sum of all sample values collected during the five-minute period, while the **Minimum** is the lowest value collected during the five-minute period.

CloudWatch supports the following statistics for metrics.

- **SampleCount** is the number of data points during the period.
- **Sum** is the sum of the values of the all data points collected during the period.
- **Average** is the value of `Sum/SampleCount` during the specified period.
- **Minimum** is the lowest value observed during the specified period.
- **Maximum** is the highest value observed during the specified period.
- **Percentile (p)** indicates the relative standing of a value in a dataset. For example, **p95** is the 95th percentile and means that 95 percent of the data within the period is lower than this value and 5 percent of the data is higher than this value. Percentiles help you get a better understanding of the distribution of your metric data.
- **Trimmed mean (TM)** is the mean of all values that are between two specified boundaries. Values outside of the boundaries are ignored when the mean is calculated. You define the boundaries as one or two numbers between 0 and 100, up to 10 decimal places. The numbers can be absolute values or percentages. For example, **tm90** calculates the average after removing the 10% of data points with the highest values. **TM(2%:98%)** calculates the average after removing the 2% lowest data points and the 2% highest data points. **TM(150:1000)** calculates the average after removing all data points that are lower than or equal to 150, or higher than 1000.
- **Interquartile mean (IQM)** is the trimmed mean of the *interquartile range*, or the middle 50% of values. It is equivalent to **TM(25%:75%)**.
- **Winsorized mean (WM)** is similar to trimmed mean. However, with winsorized mean, the values that are outside the boundary are not ignored, but instead are considered to be equal to the value at the edge of the appropriate boundary. After this normalization, the average is calculated. You define the boundaries as one or two numbers between 0 and 100, up to 10 decimal places. For example, **wm98** calculates the average while treating the 2% of the highest values to be equal to the value at the 98th percentile. **WM(10%:90%)** calculates the average while treating the highest 10% of data points to be the value of the 90% boundary, and treating the lowest 10% of data points to be the value of the 10% boundary.
- **Percentile rank (PR)** is the percentage of values that meet a fixed threshold. For example, **PR(:300)** returns the percentage of data points that have a value of 300 or less. **PR(100:2000)** returns the percentage of data points that have a value between 100 and 2000.

  Percentile rank is exclusive on the lower bound and inclusive on the upper bound.

- **Trimmed count (TC)** is the number of data points in the chosen range for a trimmed mean statistic. For example, **tc90** returns the number of data points not including any data points that fall in the highest 10% of the values. **TC(0.005:0.030)** returns the number of data points with values between 0.005 (exclusive) and 0.030 (inclusive).

  Trimmed count can return a decimal value instead of an integer. This is because it is an interpolated, approximate value and can give fractional results.
- **Trimmed sum (TS)** is the sum of the values of data points in a chosen range for a trimmed mean statistic. It is equivalent to (Trimmed Mean) * (Trimmed count). For example, **ts90** returns the sum of the data points not including any data points that fall in the highest 10% of the values. **TS(80%:)** returns the sum of the data point values, not including any data points with values in the lowest 80% of the range of values.

> **Note**
>
> For Trimmed Mean, Trimmed Count, Trimmed Sum, and Winsorized Mean, if you define two boundaries as fixed values instead of percentages, the calculation includes values equal to the higher boundary, but does not include values equal to the lower boundary.

Syntax

For Trimmed Mean, Trimmed Count, Trimmed Sum, and Winsorized Mean, the following syntax rules apply:

- Using parentheses with one or two numbers with percent signs defines the boundaries to use as the values in the data set that fall in between the two percentiles that you specify. For example, **TM(10%:90%)** uses only the values between the 10th and 90th percentiles. **TM(:95%)** uses the values from the lowest end of the data set up to the 95th percentile, ignoring the 5% of data points with the highest values.
- Using parenthesis with one or two numbers without percent signs defines the boundaries to use as the values in the data set that fall in between the explicit values that you specify. For example, **TC(80:500)** uses only the values that are between 80 (exclusive) and 500 (inclusive). **TC(:0.5)** uses only the values that equal 0.5 or are lower.
- Using one number without parentheses calculates using percentages, ignoring data points that are higher than the specified percentile. For example, **tm99** calculates the mean while ignoring the 1% of the data points with the highest value. It is the same as **TM(:99%)**.

- Trimmed mean, Trimmed Count, Trimmed Sum, and Winsorized Mean can all be abbreviated using uppercase letters when specifying a range, such as **TM(5%:95%)**, **TM(100:200)**, or **TM(:95%)**. They can only be abbreviated using lowercase letters when you specifying only one number, such as **tm99**.

Statistics use cases

- **Trimmed mean** is most useful for metrics with a large sample size, such as webpage latency. For example, **tm99** disregards extreme high outliers that could result from network problems or human errors, to give a more accurate number for the average latency of typical requests. Similarly, **TM(10%:)** disregards the lowest 10% of latency values, such as those resulting from cache hits. And **TM(10%:99%)** excludes both of these types of outliers. We recommend that you use trimmed mean for monitoring latency.
- It is a good idea to keep watch on trimmed count whenever you are using trimmed mean, to make sure that the number of values being used in your trimmed mean calculations are enough to be statistically significant.
- Percentile rank enables you to put values into "bins" of ranges, and you can use this to manually create a histogram. To do this, break your values down into various bins, such as **PR(:1)**, **PR(1:5)**, **PR(5:10)**, and **PR(10:)**. Put each of these bins into a visualization as bar charts, and you have a histogram.

  Percentile rank is exclusive on the lower bound and inclusive on the upper bound.

Percentiles versus trimmed mean

A percentile such as **p99** and a trimmed mean such as **tm99** measure similar, but not identical values. Both **p99** and **tm99** ignore the 1% of the data points with the highest values, which are considered outliers. After that, **p99** is the maximum value of the remaining 99%, while **tm99** is the *average* of the remaining 99%. If you are looking at the latency of web requests, **p99** tells you the worst customer experience, ignoring outliers, while **tm99** tells you the average customer experience, ignoring outliers.

Trimmed mean is a good latency statistic to watch if you are looking to optimize your customer experience.

Requirements to use percentiles, trimmed mean, and some other statistics

CloudWatch needs raw data points to calculate the following statistics:

- Percentiles
- Trimmed mean
- Interquartile mean
- Winsorized mean
- Trimmed sum
- Trimmed count
- Percentile rank

If you publish data for a custom statistics using a statistic set instead of raw data, you can retrieve these types of statistics for this data only if one of the following conditions is true:

- The SampleCount value of the statistic set is 1 and Min, Max, and Sum are all equal.
- The Min and Max are equal, and Sum is equal to Min multiplied by SampleCount.

The following Amazon services include metrics that support these types of statistics.

- API Gateway
- Application Load Balancer
- Amazon EC2
- Elastic Load Balancing
- Kinesis
- Amazon RDS

Additionally, these type of statistics are not available for metrics when any of the metric values are negative numbers.

The following examples show you how to get statistics for the CloudWatch metrics for your resources, such as your EC2 instances.

Examples

- Get statistics for a specific resource

- Aggregate statistics across resources
- Aggregate statistics by Auto Scaling group
- Aggregate statistics by Amazon Machine Image (AMI)

Get statistics for a specific resource

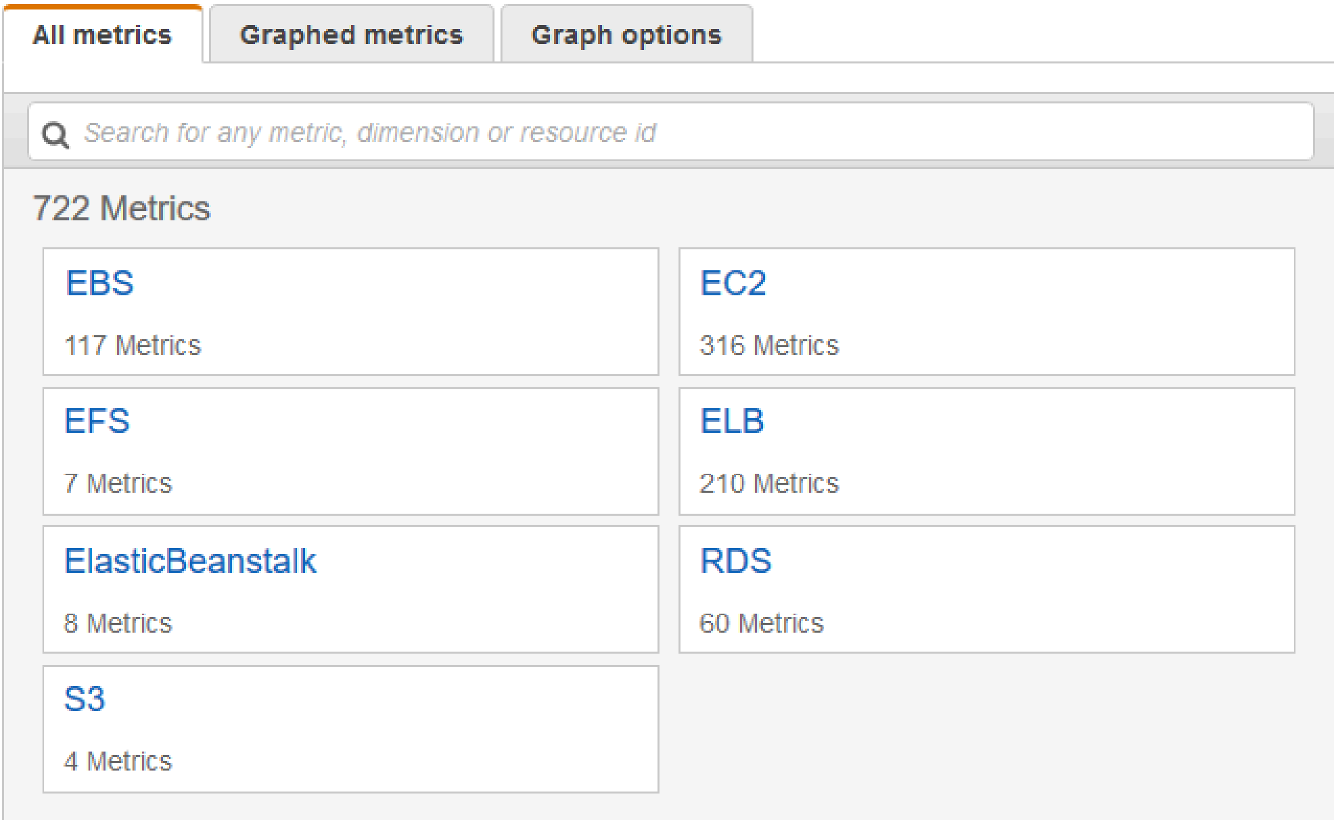
The following example shows you how to determine the maximum CPU utilization of a specific EC2 instance.

Requirements

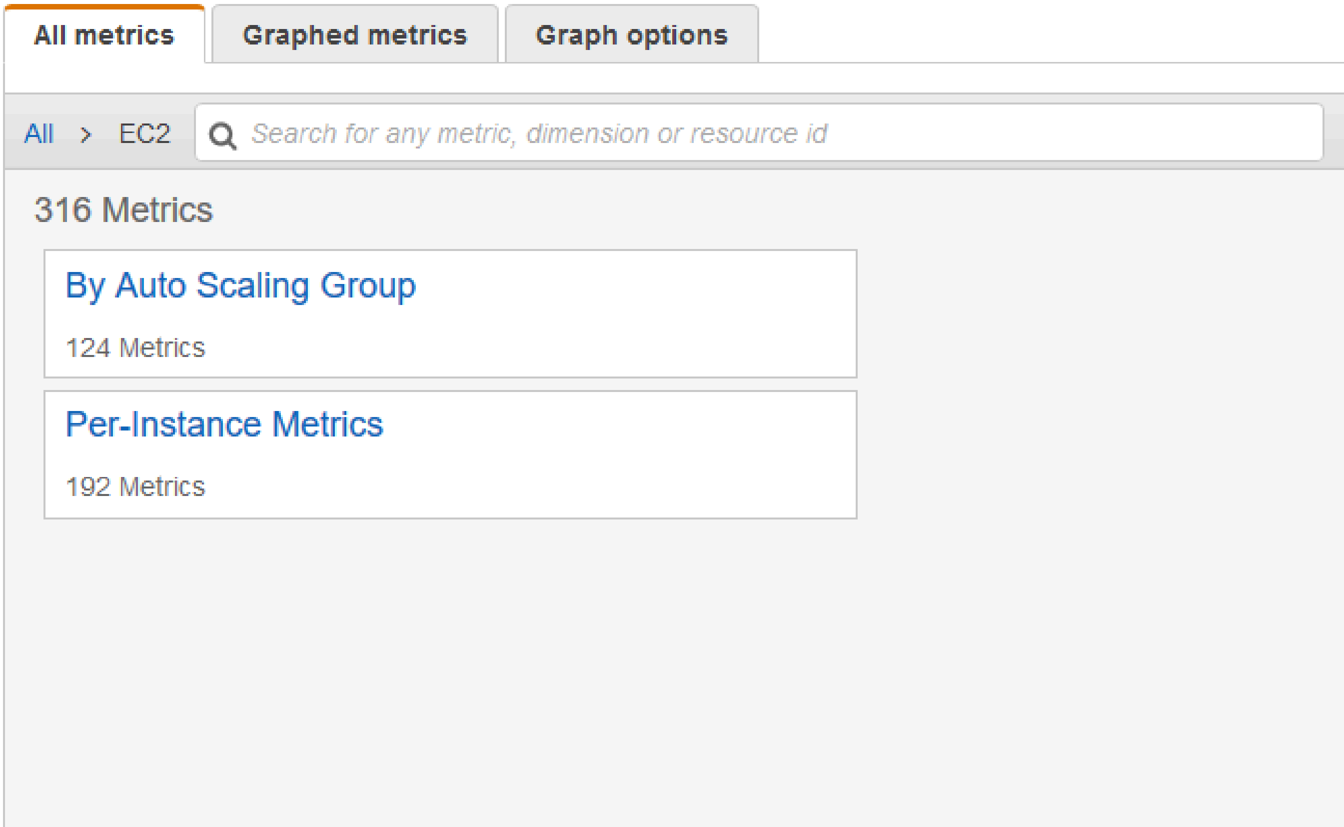
- You must have the ID of the instance. You can get the instance ID using the Amazon EC2 console or the describe-instances command.
- By default, basic monitoring is enabled, but you can enable detailed monitoring. For more information, see Enable or Disable Detailed Monitoring for Your Instances in the *Amazon EC2 User Guide*.

To display the average CPU utilization for a specific instance using the console

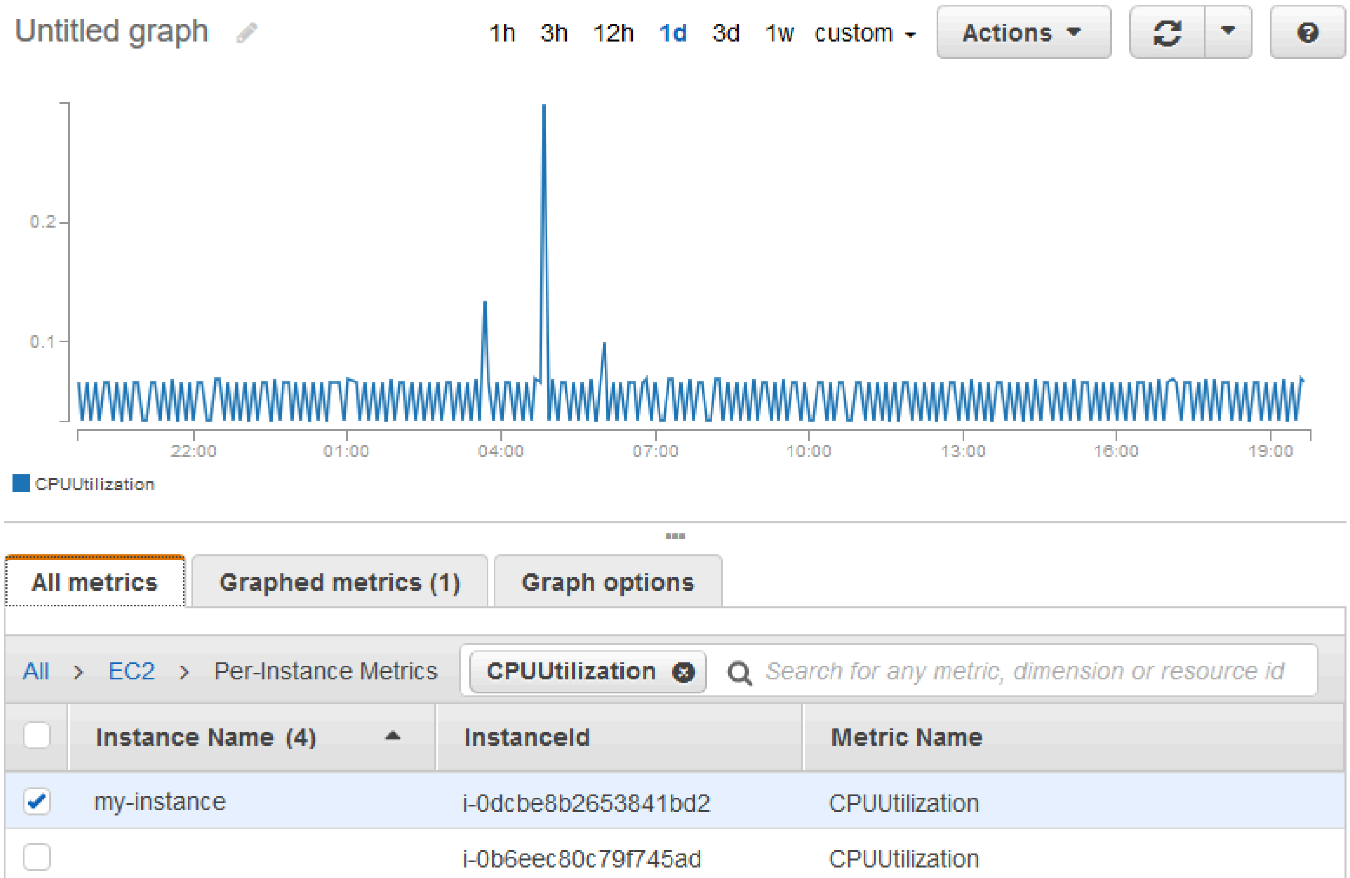
1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Metrics**, **All metrics**.
3. Select the **EC2** metric namespace.



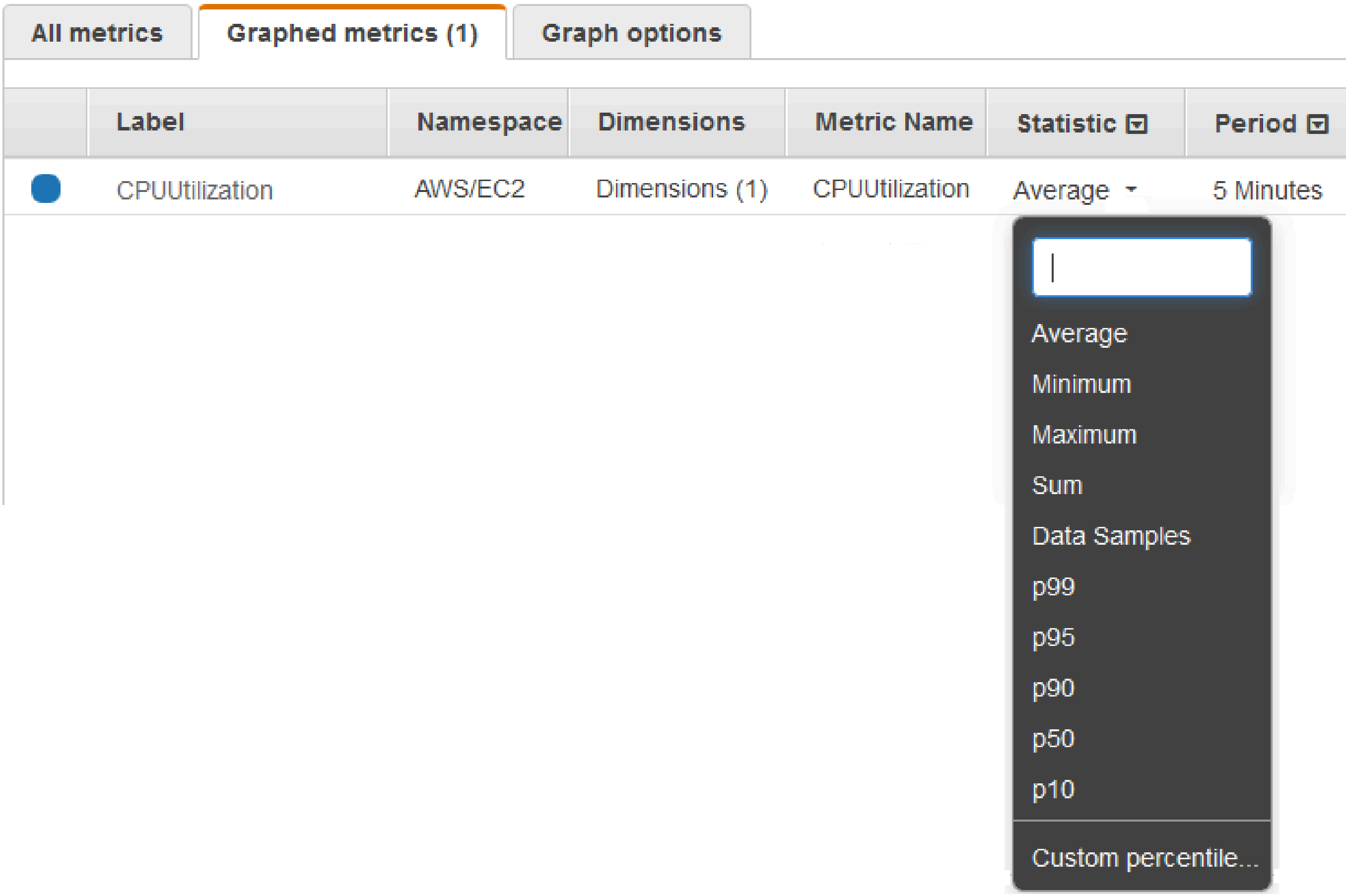
4. Select the **Per-Instance Metrics** dimension.



5. In the search field, enter **`CPUUtilization`** and press Enter. Select the row for the specific instance, which displays a graph for the `CPUUtilization` metric for the instance. To change the name of the graph, choose the pencil icon. To change the time range, select one of the predefined values or choose **custom**.



6. To change the statistic, choose the **Graphed metrics** tab. Choose the column heading or an individual value and then choose one of the statistics or predefined percentiles, or specify a custom percentile (for example, **`p99.999`**).



7. To change the period, choose the **Graphed metrics** tab. Choose the column heading or an individual value, and then choose a different value.

To get the CPU utilization per EC2 instance using the Amazon CLI

Use the get-metric-statistics command as follows to get the `CPUUtilization` metric for the specified instance.

```
aws cloudwatch get-metric-statistics --namespace AWS/EC2 --metric-name CPUUtilization \
--dimensions Name=InstanceId,Value=i-1234567890abcdef0 --statistics Maximum \
--start-time 2016-10-18T23:18:00 --end-time 2016-10-19T23:18:00 --period 360
```

The returned statistics are 6-minute values for the requested 24-hour time interval. Each value represents the maximum CPU utilization percentage for the specified instance for a particular 6-minute time period. The data points aren't returned in chronological order. The following shows the beginning of the example output (the full output includes data points for every 6 minutes of the 24-hour period).

```
{
    "Datapoints": [
        {
            "Timestamp": "2016-10-19T00:18:00Z",
            "Maximum": 0.33000000000000002,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2016-10-19T03:18:00Z",
            "Maximum": 99.670000000000002,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2016-10-19T07:18:00Z",
            "Maximum": 0.34000000000000002,
            "Unit": "Percent"
        },
        ...
    ],
    "Label": "CPUUtilization"
}
```

Aggregate statistics across resources

You can aggregate the metrics for Amazon resources across multiple resources. Metrics are completely separate between Regions, but you can use metric math to aggregate similar metrics across Regions. For more information, see Using math expressions with CloudWatch metrics.

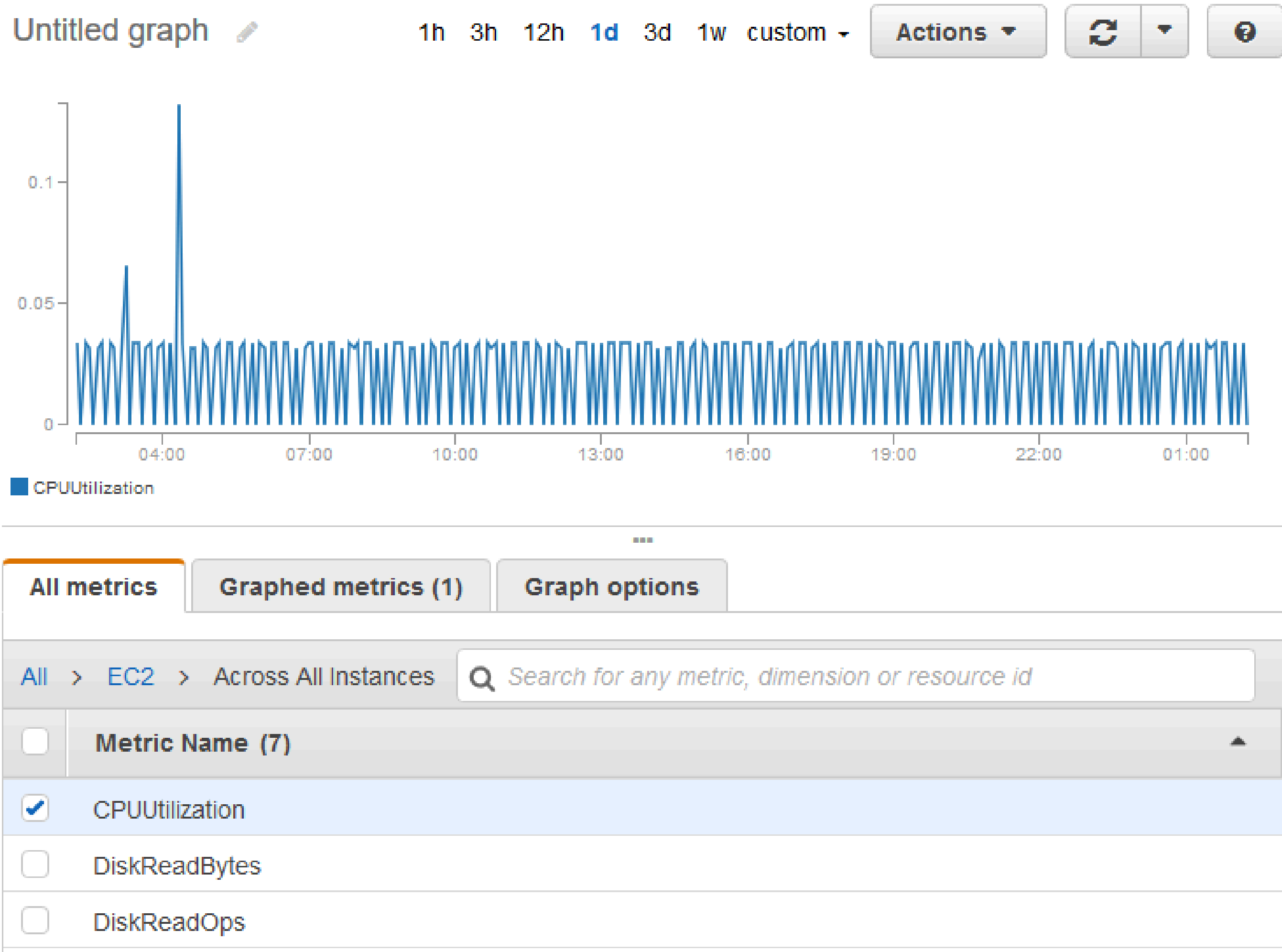
For example, you can aggregate statistics for your EC2 instances that have detailed monitoring enabled. Instances that use basic monitoring aren't included. Therefore, you must enable detailed monitoring (at an additional charge), which provides data in 1-minute periods. For more information, see Enable or Disable Detailed Monitoring for Your Instances in the *Amazon EC2 User Guide*.

This example shows you how to get the average CPU usage for your EC2 instances. Because no dimension is specified, CloudWatch returns statistics for all dimensions in the `AWS/EC2` namespace. To get statistics for other metrics, see Amazon services that publish CloudWatch metrics.

> **Important**
>
> This technique for retrieving all dimensions across an Amazon namespace doesn't work for custom namespaces that you publish to CloudWatch. With custom namespaces, you must specify the complete set of dimensions that are associated with any given data point to retrieve statistics that include the data point.

To display average CPU utilization for your EC2 instances

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Metrics**, **All metrics**.
3. Choose the **EC2** namespace and choose **Across All Instances**.
4. Select the row that contains `CPUUtilization`, which displays a graph for the metric for all your EC2 instances. To change the name of the graph, choose the pencil icon. To change the time range, select one of the predefined values or choose **custom**.



5. To change the statistic, choose the **Graphed metrics** tab. Choose the column heading or an individual value and then choose one of the statistics or predefined percentiles, or specify a custom percentile (for example, **p95.45**).

6. To change the period, choose the **Graphed metrics** tab. Choose the column heading or an individual value and then choose a different value.

To get average CPU utilization across your EC2 instances using the Amazon CLI

Use the get-metric-statistics command as follows:

```
aws cloudwatch get-metric-statistics --namespace AWS/EC2 --metric-name CPUUtilization
 --statistics "Average" "SampleCount" \
--start-time 2016-10-11T23:18:00 --end-time 2016-10-12T23:18:00 --period 3600
```

The following is example output:

```
{
    "Datapoints": [
        {
            "SampleCount": 238.0,
            "Timestamp": "2016-10-12T07:18:00Z",
            "Average": 0.038235294117647062,
            "Unit": "Percent"
        },
        {
            "SampleCount": 240.0,
            "Timestamp": "2016-10-12T09:18:00Z",
            "Average": 0.16670833333333332,
            "Unit": "Percent"
        },
        {
            "SampleCount": 238.0,
            "Timestamp": "2016-10-11T23:18:00Z",
            "Average": 0.041596638655462197,
            "Unit": "Percent"
        },
        ...
    ],
    "Label": "CPUUtilization"
}
```


Aggregate statistics by Auto Scaling group

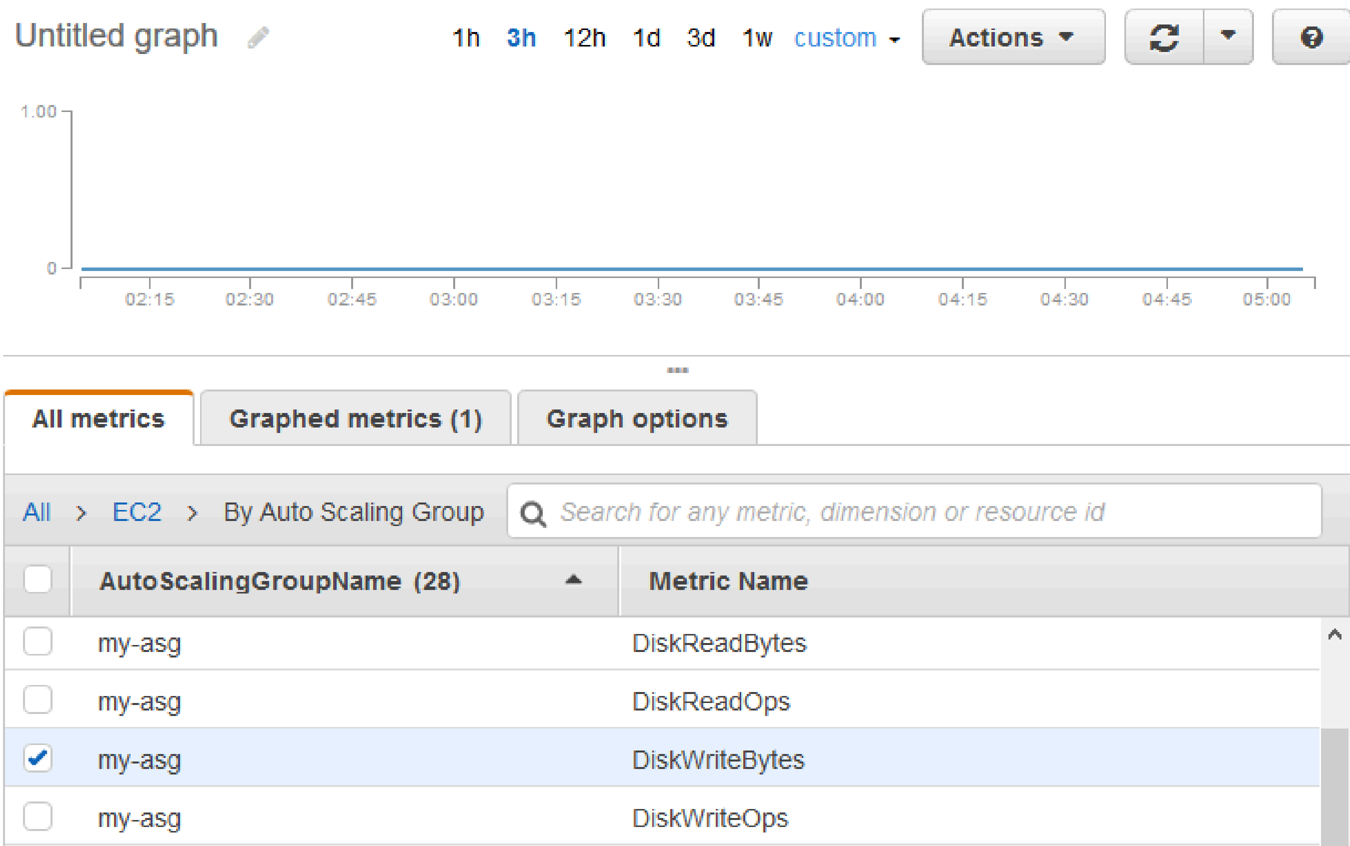
You can aggregate statistics for the EC2 instances in an Auto Scaling group. Metrics are completely separate between Regions, but you can use CloudWatch metric math to aggregate and transform metrics from multiple Regions. You can also use the cross-account dashboard to perform metric math on metrics from different accounts.

This example shows you how to get the total bytes written to disk for one Auto Scaling group. The total is computed for 1-minute periods for a 24-hour interval across all EC2 instances in the specified Auto Scaling group.

To display DiskWriteBytes for the instances in an Auto Scaling group using the console

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Metrics**, **All metrics**.
3. Choose the **EC2** namespace and then choose **By Auto Scaling Group**.

4. Select the row for the **DiskWriteBytes** metric and the specific Auto Scaling group, which displays a graph for the metric for the instances in the Auto Scaling group. To change the name of the graph, choose the pencil icon. To change the time range, select one of the predefined values or choose **custom**.



5. To change the statistic, choose the **Graphed metrics** tab. Choose the column heading or an individual value and then choose one of the statistics or predefined percentiles, or specify a custom percentile (for example, **p95.45**).
6. To change the period, choose the **Graphed metrics** tab. Choose the column heading or an individual value and then choose a different value.

To get DiskWriteBytes for the instances in an Auto Scaling group using the Amazon CLI

Use the get-metric-statistics command as follows.

```
aws cloudwatch get-metric-statistics --namespace AWS/EC2 --metric-name DiskWriteBytes
--dimensions Name=AutoScalingGroupName,Value=my-asg --statistics "Sum" "SampleCount" \
--start-time 2016-10-16T23:18:00 --end-time 2016-10-18T23:18:00 --period 360
```

The following is example output.

```
{
    "Datapoints": [
        {
            "SampleCount": 18.0,
            "Timestamp": "2016-10-19T21:36:00Z",
            "Sum": 0.0,
            "Unit": "Bytes"
        },
        {
            "SampleCount": 5.0,
            "Timestamp": "2016-10-19T21:42:00Z",
            "Sum": 0.0,
            "Unit": "Bytes"
        }
    ],
    "Label": "DiskWriteBytes"
}
```

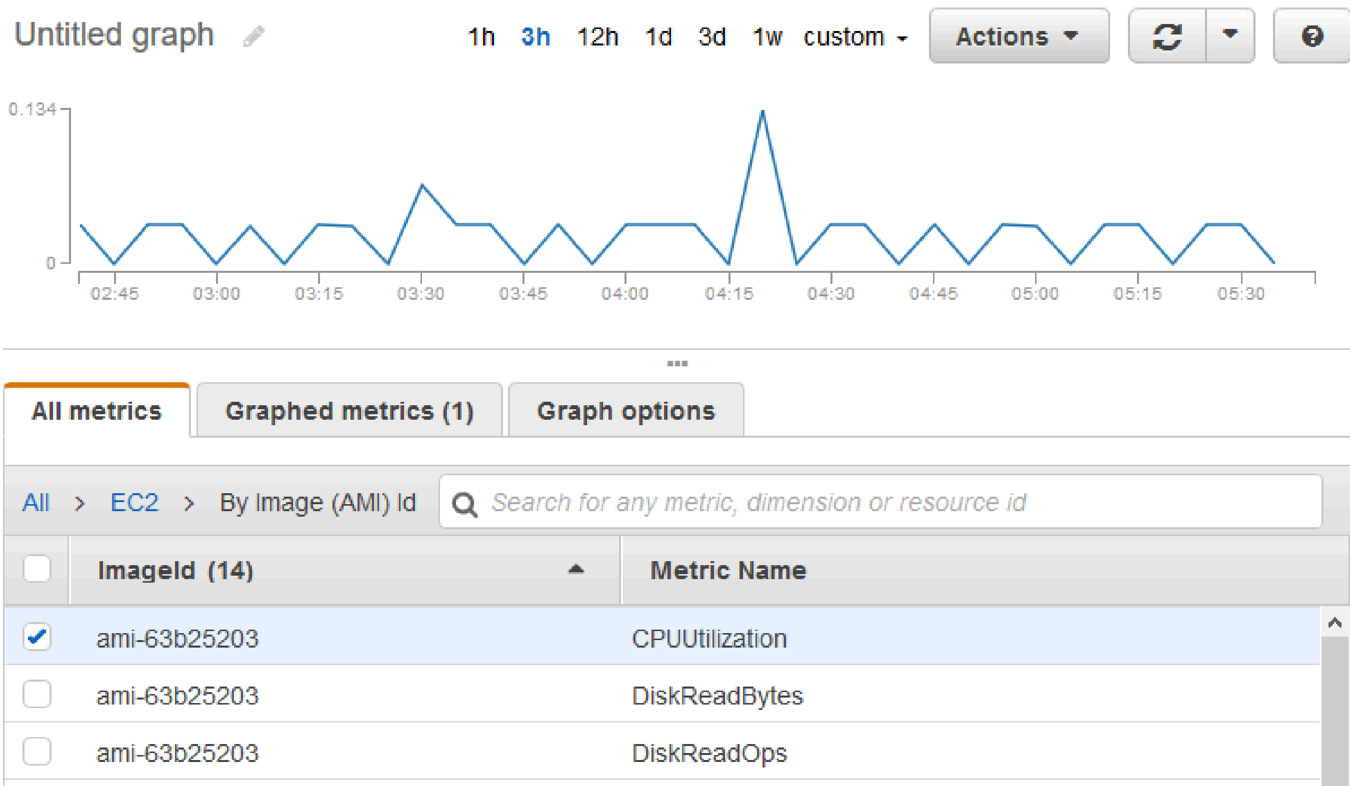
Aggregate statistics by Amazon Machine Image (AMI)

You can aggregate statistics for the EC2 instances that have detailed monitoring enabled. Instances that use basic monitoring aren't included. For more information, see Enable or Disable Detailed Monitoring for Your Instances in the *Amazon EC2 User Guide*.

This example shows you how to determine average CPU utilization for all instances that use the specified AMI. The average is over 60-second time intervals for a one-day period.

To display the average CPU utilization by AMI using the console

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Metrics**, **All metrics**.
3. Choose the **EC2** namespace and then choose **By Image (AMI) Id**.
4. Select the row for the `CPUUtilization` metric and the specific AMI, which displays a graph for the metric for the specified AMI. To change the name of the graph, choose the pencil icon. To change the time range, select one of the predefined values or choose **custom**.



5. To change the statistic, choose the **Graphed metrics** tab. Choose the column heading or an individual value and then choose one of the statistics or predefined percentiles, or specify a custom percentile (for example, **`p95.45`**).
6. To change the period, choose the **Graphed metrics** tab. Choose the column heading or an individual value and then choose a different value.

To get the average CPU utilization by AMI using the Amazon CLI

Use the get-metric-statistics command as follows.

```
aws cloudwatch get-metric-statistics --namespace AWS/EC2 --metric-name CPUUtilization \
--dimensions Name=ImageId,Value=ami-3c47a355 --statistics Average \
--start-time 2016-10-10T00:00:00 --end-time 2016-10-11T00:00:00 --period 3600
```

The operation returns statistics that are one-hour values for the one-day interval. Each value represents an average CPU utilization percentage for EC2 instances running the specified AMI. The following is example output.

```
{
    "Datapoints": [
```

```
        {
            "Timestamp": "2016-10-10T07:00:00Z",
            "Average": 0.04100000000000009,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2016-10-10T14:00:00Z",
            "Average": 0.079579831932773085,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2016-10-10T06:00:00Z",
            "Average": 0.036000000000000011,
            "Unit": "Percent"
        },
        ...
    ],
    "Label": "CPUUtilization"
}
```

Publish custom metrics

You can publish your own metrics to CloudWatch using the Amazon CLI or an API. You can view statistical graphs of your published metrics with the Amazon Web Services Management Console.

CloudWatch stores data about a metric as a series of data points. Each data point has an associated time stamp. You can even publish an aggregated set of data points called a *statistic set*.

Topics

- High-resolution metrics
- Use dimensions
- Publish single data points
- Publish statistic sets
- Publish the value zero
- Stop publishing metrics

High-resolution metrics

Each metric is one of the following:

- Standard resolution, with data having a one-minute granularity
- High resolution, with data at a granularity of one second

Metrics produced by Amazon services are standard resolution by default. When you publish a custom metric, you can define it as either standard resolution or high resolution. When you publish a high-resolution metric, CloudWatch stores it with a resolution of 1 second, and you can read and retrieve it with a period of 1 second, 5 seconds, 10 seconds, 30 seconds, or any multiple of 60 seconds.

High-resolution metrics can give you more immediate insight into your application's sub-minute activity. Keep in mind that every `PutMetricData` call for a custom metric is charged, so calling `PutMetricData` more often on a high-resolution metric can lead to higher charges. For more information about CloudWatch pricing, see [Amazon CloudWatch Pricing](#).

If you set an alarm on a high-resolution metric, you can specify a high-resolution alarm with a period of 10 seconds or 30 seconds, or you can set a regular alarm with a period of any multiple of 60 seconds. There is a higher charge for high-resolution alarms with a period of 10 or 30 seconds.

Use dimensions

In custom metrics, the `--dimensions` parameter is common. A dimension further clarifies what the metric is and what data it stores. You can have up to 30 dimensions assigned to one metric, and each dimension is defined by a name and value pair.

How you specify a dimension is different when you use different commands. With put-metric-data, you specify each dimension as *MyName*=*MyValue*, and with get-metric-statistics or put-metric-alarm you use the format `Name=`*MyName*, `Value=`*MyValue*. For example, the following command publishes a `Buffers` metric with two dimensions named `InstanceId` and `InstanceType`.

```
aws cloudwatch put-metric-data --metric-name Buffers --namespace MyNameSpace --unit
 Bytes --value 231434333 --dimensions InstanceId=1-23456789,InstanceType=m1.small
```

This command retrieves statistics for that same metric. Separate the Name and Value parts of a single dimension with commas, but if you have multiple dimensions, use a space between one dimension and the next.

```
aws cloudwatch get-metric-statistics --metric-name Buffers --namespace MyNameSpace --
dimensions Name=InstanceId,Value=1-23456789 Name=InstanceType,Value=m1.small --start-
```

```
time 2016-10-15T04:00:00Z --end-time 2016-10-19T07:00:00Z --statistics Average --period
 60
```

If a single metric includes multiple dimensions, you must specify a value for every defined dimension when you use [get-metric-statistics](). For example, the Amazon S3 metric `BucketSizeBytes` includes the dimensions `BucketName` and `StorageType`, so you must specify both dimensions with [get-metric-statistics]().

```
aws cloudwatch get-metric-statistics --metric-name BucketSizeBytes --start-time
 2017-01-23T14:23:00Z --end-time 2017-01-26T19:30:00Z --period 3600 --namespace
 AWS/S3 --statistics Maximum --dimensions Name=BucketName,Value=amzn-s3-demo-bucket
 Name=StorageType,Value=StandardStorage --output table
```

To see what dimensions are defined for a metric, use the [list-metrics]() command.

Publish single data points

To publish a single data point for a new or existing metric, use the [put-metric-data]() command with one value and time stamp. For example, the following actions each publish one data point.

```
aws cloudwatch put-metric-data --metric-name PageViewCount --namespace MyService --
value 2 --timestamp 2016-10-20T12:00:00.000Z
aws cloudwatch put-metric-data --metric-name PageViewCount --namespace MyService --
value 4 --timestamp 2016-10-20T12:00:01.000Z
aws cloudwatch put-metric-data --metric-name PageViewCount --namespace MyService --
value 5 --timestamp 2016-10-20T12:00:02.000Z
```

If you call this command with a new metric name, CloudWatch creates a metric for you. Otherwise, CloudWatch associates your data with the existing metric that you specified.

> **Note**
>
> When you create a metric, it can take up to 2 minutes before you can retrieve statistics for the new metric using the [get-metric-statistics]() command. However, it can take up to 15 minutes before the new metric appears in the list of metrics retrieved using the [list-metrics]() command.

Although you can publish data points with time stamps as granular as one-thousandth of a second, CloudWatch aggregates the data to a minimum granularity of 1 second. CloudWatch records the average (sum of all items divided by number of items) of the values received for each period, as well as the number of samples, maximum value, and minimum value for the same time period. For example, the PageViewCount metric from the previous examples contains three data points with time stamps just seconds apart. If you have your period set to 1 minute, CloudWatch aggregates the three data points because they all have time stamps within a 1-minute period.

You can use the **get-metric-statistics** command to retrieve statistics based on the data points that you published.

```
aws cloudwatch get-metric-statistics --namespace MyService --metric-name PageViewCount
 \
--statistics "Sum" "Maximum" "Minimum" "Average" "SampleCount" \
--start-time 2016-10-20T12:00:00.000Z --end-time 2016-10-20T12:05:00.000Z --period 60
```

The following is example output.

```
{
    "Datapoints": [
        {
            "SampleCount": 3.0,
            "Timestamp": "2016-10-20T12:00:00Z",
            "Average": 3.6666666666666665,
            "Maximum": 5.0,
            "Minimum": 2.0,
            "Sum": 11.0,
            "Unit": "None"
        }
    ],
    "Label": "PageViewCount"
}
```

Publish statistic sets

You can aggregate your data before you publish to CloudWatch. When you have multiple data points per minute, aggregating data minimizes the number of calls to **put-metric-data**. For example, instead of calling **put-metric-data** multiple times for three data points that are within 3 seconds of each other, you can aggregate the data into a statistic set that you publish with one call, using the `--statistic-values` parameter.

```
aws cloudwatch put-metric-data --metric-name PageViewCount --namespace MyService
 --statistic-values Sum=11,Minimum=2,Maximum=5,SampleCount=3 --
timestamp 2016-10-14T12:00:00.000Z
```

CloudWatch needs raw data points to calculate percentiles. If you publish data using a statistic set instead, you can't retrieve percentile statistics for this data unless one of the following conditions is true:

- The `SampleCount` of the statistic set is 1
- The `Minimum` and the `Maximum` of the statistic set are equal

Publish the value zero

When your data is more sporadic and you have periods that have no associated data, you can choose to publish the value zero (`0`) for that period or no value at all. If you use periodic calls to `PutMetricData` to monitor the health of your application, you might want to publish zero instead of no value. For example, you can set a CloudWatch alarm to notify you if your application fails to publish metrics every five minutes. You want such an application to publish zeros for periods with no associated data.

You might also publish zeros if you want to track the total number of data points or if you want statistics such as minimum and average to include data points with the value 0.

Stop publishing metrics

To stop publishing custom metrics to CloudWatch, change your application's or service's code to stop using **PutMetricData**. CloudWatch doesn't pull metrics from applications, it only receives what is pushed to it, so to stop publishing your metrics you must stop them at the source.

Using Amazon CloudWatch alarms

You can create alarms that watch metrics and send notifications or automatically make changes to the resources you are monitoring when a threshold is breached. For example, you can monitor the CPU usage and disk reads and writes of your Amazon EC2 instances and then use that data to determine whether you should launch additional instances to handle increased load. You can also use this data to stop under-used instances to save money.

You can create both *metric* and *composite* alarms in Amazon CloudWatch.

- A *metric alarm* watches a single CloudWatch metric or the result of a math expression based on CloudWatch metrics. The alarm performs one or more actions based on the value of the metric or expression relative to a threshold over a number of time periods. The action can be sending a notification to an Amazon SNS topic, performing an Amazon EC2 action or an Amazon EC2 Auto Scaling action, starting an investigation in Amazon Q Developer operational investigations, or creating an OpsItem or incident in Systems Manager.
- A *composite alarm* includes a rule expression that takes into account the alarm states of other alarms that you have created. The composite alarm goes into ALARM state only if all conditions of the rule are met. The alarms specified in a composite alarm's rule expression can include metric alarms and other composite alarms.

  Using composite alarms can reduce alarm noise. You can create multiple metric alarms, and also create a composite alarm and set up alerts only for the composite alarm. For example, a composite might go into ALARM state only when all of the underlying metric alarms are in ALARM state.

  Composite alarms can send Amazon SNS notifications when they change state, and can create investigations, Systems Manager OpsItems, or incidents when they go into ALARM state, but can't perform EC2 actions or Auto Scaling actions.

> **Note**
>
> You can create as many alarms as you want in your Amazon account.

You can add alarms to dashboards, so you can monitor and receive alerts about your Amazon resources and applications across multiple regions. After you add an alarm to a dashboard, the alarm turns gray when it's in the `INSUFFICIENT_DATA` state and red when it's in the `ALARM` state. The alarm is shown with no color when it's in the `OK` state.

You also can favorite recently visited alarms from the *Favorites and recents* option in the CloudWatch console navigation pane. The *Favorites and recents* option has columns for your favorited alarms and recently visited alarms.

An alarm invokes actions only when the alarm changes state. The exception is for alarms with Auto Scaling actions. For Auto Scaling actions, the alarm continues to invoke the action once per minute that the alarm remains in the new state.

An alarm can watch a metric in the same account. If you have enabled cross-account functionality in your CloudWatch console, you can also create alarms that watch metrics in other Amazon accounts. Creating cross-account composite alarms is not supported. Creating cross-account alarms that use math expressions is supported, except that the `ANOMALY_DETECTION_BAND`, `INSIGHT_RULE`, and `SERVICE_QUOTA` functions are not supported for cross-account alarms.

Note

CloudWatch doesn't test or validate the actions that you specify, nor does it detect any Amazon EC2 Auto Scaling or Amazon SNS errors resulting from an attempt to invoke nonexistent actions. Make sure that your alarm actions exist.

Metric alarm states

A metric alarm has the following possible states:

- `OK` – The metric or expression is within the defined threshold.
- `ALARM` – The metric or expression is outside of the defined threshold.
- `INSUFFICIENT_DATA` – The alarm has just started, the metric is not available, or not enough data is available for the metric to determine the alarm state.

Evaluating an alarm

When you create an alarm, you specify three settings to enable CloudWatch to evaluate when to change the alarm state:

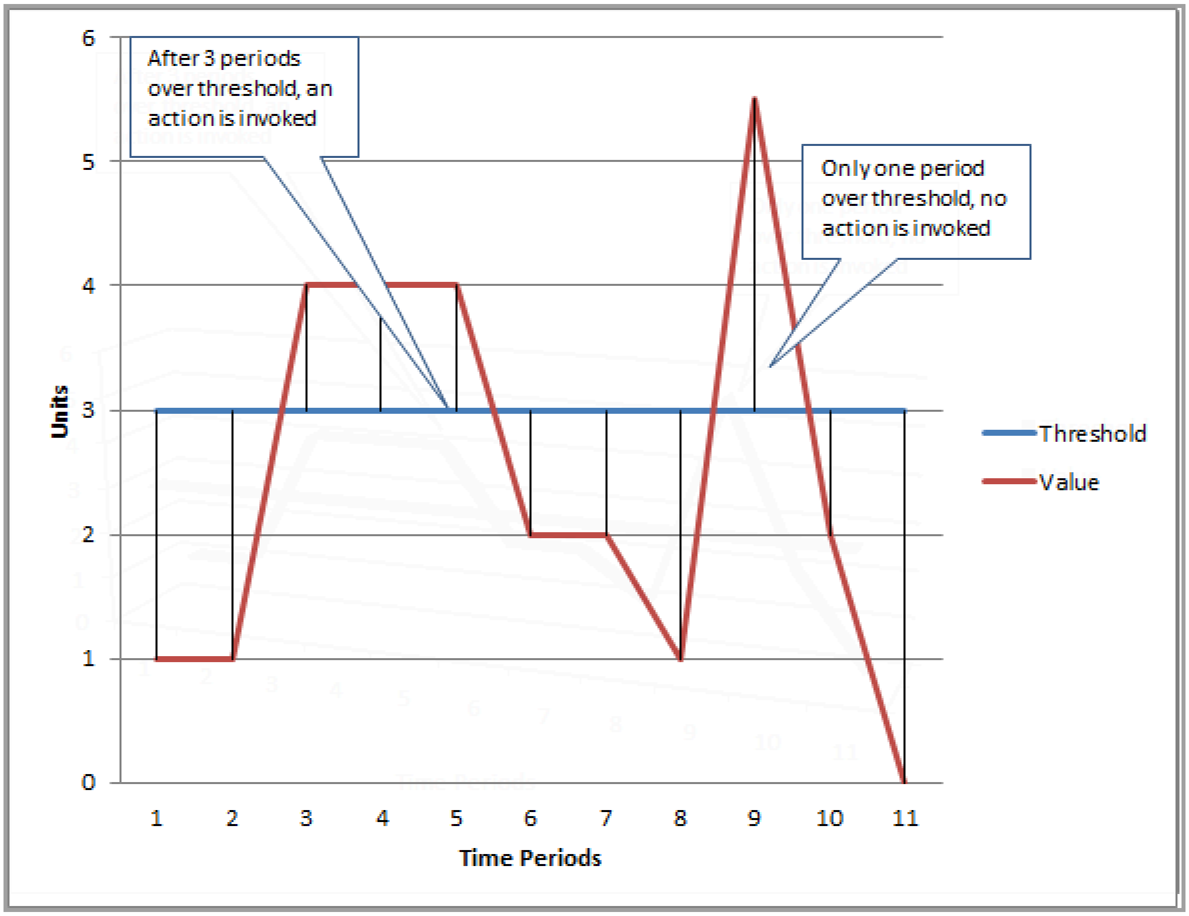
- **Period** is the length of time to use to evaluate the metric or expression to create each individual data point for an alarm. It is expressed in seconds.
- **Evaluation Periods** is the number of the most recent periods, or data points, to evaluate when determining alarm state.
- **Datapoints to Alarm** is the number of data points within the Evaluation Periods that must be breaching to cause the alarm to go to the `ALARM` state. The breaching data points don't have to be consecutive, but they must all be within the last number of data points equal to **Evaluation Period**.

For any period of one minute or longer, an alarm is evaluated every minute and the evaluation is based on the window of time defined by the **Period** and **Evaluation Periods**. For example, if the **Period** is 5 minutes (300 seconds) and **Evaluation Periods** is 1, then at the end of minute 5 the alarm evaluates based on data from minutes 1 to 5. Then at the end of minute 6, the alarm is evaluated based on the data from minutes 2 to 6.

If the alarm period is 10 seconds, 20 seconds, or 30 seconds, the alarm is evaluated every 10 seconds.

If the number of evaluation periods multiplied by the length of each evaluation period exceeds one day, the alarm is evaluated once per hour. For more details about how these multi-day alarms are evaluated, see the example at the end of this section.

In the following figure, the alarm threshold for a metric alarm is set to three units. Both **Evaluation Period** and **Datapoints to Alarm** are 3. That is, when all existing data points in the most recent three consecutive periods are above the threshold, the alarm goes to `ALARM` state. In the figure, this happens in the third through fifth time periods. At period six, the value dips below the threshold, so one of the periods being evaluated is not breaching, and the alarm state changes back to `OK`. During the ninth time period, the threshold is breached again, but for only one period. Consequently, the alarm state remains `OK`.



When you configure **Evaluation Periods** and **Datapoints to Alarm** as different values, you're setting an "M out of N" alarm. **Datapoints to Alarm** is ("M") and **Evaluation Periods** is ("N"). The evaluation interval is the number of evaluation periods multiplied by the period length. For example, if you configure 4 out of 5 data points with a period of 1 minute, the evaluation interval is 5 minutes. If you configure 3 out of 3 data points with a period of 10 minutes, the evaluation interval is 30 minutes.

Note

If data points are missing soon after you create an alarm, and the metric was being reported to CloudWatch before you created the alarm, CloudWatch retrieves the most recent data points from before the alarm was created when evaluating the alarm.

Example of evaluating a multi-day alarm

An alarm is a multi-day alarm if the number of evaluation periods multiplied by the length of each evaluation period exceeds one day. Multi-day alarms are evaluated once per hour. When multi-day alarms are evaluated, CloudWatch takes into account only the metrics up to the current hour at the :00 minute when evaluating.

For example, consider an alarm that monitors a job that runs every 3 days at 10:00.

1. At 10:02, the job fails
2. At 10:03, the alarm evaluates and stays in `OK` state, because the evaluation considers data only up to 10:00.
3. At 11:03, the alarm considers data up to 11:00 and goes into `ALARM` state.
4. At 11:43, you correct the error and the job now runs successfully.
5. At 12:03, the alarm evaluates again, sees the successful job, and returns to `OK` state.

Alarm actions

You can specify what actions an alarm takes when it changes state between the OK, ALARM, and INSUFFICIENT_DATA states.

Most actions can be set for the transition into each of the three states. Except for Auto Scaling actions, the actions happen only on state transitions, and are not performed again if the condition persists for hours or days. You can use the fact that multiple actions are allowed for an alarm to send an email when a threshold is breached, and then another when the breaching condition ends. This helps you verify that your scaling or recovery actions are triggered when expected and are working as desired.

The following are supported as alarm actions.

- Notify one or more subscribers by **using an Amazon Simple Notification Service topic**. Subscribers can be applications as well as persons. For more information about Amazon SNS, see [What is Amazon SNS?](#).
- **Invoke a Lambda function.** This is the easiest way for you to automate custom actions on alarm state changes.

- Alarms based on EC2 metrics can also **perform EC2 actions**, such as stopping, terminating, rebooting, or recovering an EC2 instance. For more information, see Create alarms to stop, terminate, reboot, or recover an EC2 instance.
- Alarms can perform actions to **scale an Auto Scaling group**. For more information, see Step and simple scaling policies for Amazon EC2 Auto Scaling.
- Alarms can **create OpsItems in Systems Manager Ops Center or create incidents in Amazon Systems Manager Incident Manager**. These actions are performed only when the alarm goes into ALARM state. For more information, see Configuring CloudWatch to create OpsItems from alarms and Incident creation.
- An alarm can start an investigation when it goes into ALARM state. For more information about Amazon Q Developer operational investigations, see Amazon Q Developer operational investigations (Preview).

Alarms also emit events to Amazon EventBridge when they change state, and you can set up Amazon EventBridge to trigger other actions for these state changes. For more information, see What is Amazon EventBridge?.

Lambda alarm actions

CloudWatch alarms guarantees an asynchronous invocation of the Lambda function for a given state change, except in the following cases:

- When the function doesn't exist.
- When CloudWatch is not authorized to invoke the Lambda function.

If CloudWatch can't reach the Lambda service or the message is rejected for another reason, CloudWatch retries until the invocation is successful. Lambda queues the message and handles execution retries. For more information about this execution model, including information about how Lambda handles errors, see Asynchronous invocation in the Amazon Lambda Developer Guide.

You can invoke a Lambda function in the same account, or in other Amazon accounts.

When you specify an alarm to invoke a Lambda function as an alarm action, you can choose to specify the function name, function alias, or a specific version of a function.

When you specify a Lambda function as an alarm action, you must create a resource policy for the function to allow the CloudWatch service principal to invoke the function.

One way to do this is by using the Amazon CLI, as in the following example:

```
aws lambda add-permission \
--function-name my-function-name \
--statement-id AlarmAction \
--action 'lambda:InvokeFunction' \
--principal lambda.alarms.cloudwatch.amazonaws.com \
--source-account 111122223333 \
--source-arn arn:aws:cloudwatch:us-east-1:111122223333:alarm:alarm-name
```

Alternatively, you can create a policy similar to one of the following examples and then assign it to the function.

The following example specifies the account where the alarm is located, so that only alarms in that account (111122223333) can invoke the function.

```
{
    "Version": "2012-10-17",
    "Id": "default",
    "Statement": [{
        "Sid": "AlarmAction",
        "Effect": "Allow",
        "Principal": {
            "Service": "lambda.alarms.cloudwatch.amazonaws.com"
        },
        "Action": "lambda:InvokeFunction",
        "Resource": "arn:aws:lambda:us-east-1:444455556666:function:function-name",
        "Condition": {
            "StringEquals": {
                "AWS:SourceAccount": "111122223333"
            }
        }
    }]
}
```

The following example has a narrower scope, allowing only the specified alarm in the specified account to invoke the function.

```
{
```

```
  "Version": "2012-10-17",
  "Id": "default",
  "Statement": [
    {
      "Sid": "AlarmAction",
      "Effect": "Allow",
      "Principal": {
        "Service": "lambda.alarms.cloudwatch.amazonaws.com"
      },
      "Action": "lambda:InvokeFunction",
      "Resource": "arn:aws:lambda:us-east-1:444455556666:function:function-name",
      "Condition": {
        "StringEquals": {
          "AWS:SourceAccount": "111122223333",
          "AWS:SourceArn": "arn:aws:cloudwatch:us-east-1:111122223333:alarm:alarm-name"
        }
      }
    }]
}
```

We don't recommend creating a policy that doesn't specify a source account, because such policies are vulnerable to confused deputy issues.

Event object sent from CloudWatch to Lambda

When you configure a Lambda function as an alarm action, CloudWatch delivers a JSON payload to the Lambda function when it invokes the function. This JSON payload serves as the event object for the function. You can extract data from this JSON object and use it in your function. The following is an example of an event object from a metric alarm.

```
{
  'source': 'aws.cloudwatch',
  'alarmArn': 'arn:aws:cloudwatch:us-east-1:444455556666:alarm:lambda-demo-metric-
alarm',
  'accountId': '444455556666',
  'time': '2023-08-04T12:36:15.490+0000',
  'region': 'us-east-1',
  'alarmData': {
    'alarmName': 'lambda-demo-metric-alarm',
    'state': {
      'value': 'ALARM',
      'reason': 'test',
      'timestamp': '2023-08-04T12:36:15.490+0000'
```

```
    },
    'previousState': {
      'value': 'INSUFFICIENT_DATA',
      'reason': 'Insufficient Data: 5 datapoints were unknown.',
      'reasonData':
 '{"version":"1.0","queryDate":"2023-08-04T12:31:29.591+0000","statistic":"Average","period":60
[],"threshold":5.0,"evaluatedDatapoints":[{"timestamp":"2023-08-04T12:30:00.000+0000"},
{"timestamp":"2023-08-04T12:29:00.000+0000"},
{"timestamp":"2023-08-04T12:28:00.000+0000"},
{"timestamp":"2023-08-04T12:27:00.000+0000"},
{"timestamp":"2023-08-04T12:26:00.000+0000"}]}',
      'timestamp': '2023-08-04T12:31:29.595+0000'
    },
    'configuration': {
      'description': 'Metric Alarm to test Lambda actions',
      'metrics': [
        {
          'id': '1234e046-06f0-a3da-9534-EXAMPLEe4c',
          'metricStat': {
            'metric': {
              'namespace': 'AWS/Logs',
              'name': 'CallCount',
              'dimensions': {
                'InstanceId': 'i-12345678'
              }
            },
            'period': 60,
            'stat': 'Average',
            'unit': 'Percent'
          },
          'returnData': True
        }
      ]
    }
  }
}
```

The following is an example of an event object from a composite alarm.

```
{
  'source': 'aws.cloudwatch',
  'alarmArn': 'arn:aws:cloudwatch:us-east-1:111122223333:alarm:SuppressionDemo.Main',
  'accountId': '111122223333',
```

```
  'time': '2023-08-04T12:56:46.138+0000',
  'region': 'us-east-1',
  'alarmData': {
    'alarmName': 'CompositeDemo.Main',
    'state': {
      'value': 'ALARM',
      'reason': 'arn:aws:cloudwatch:us-
east-1:111122223333:alarm:CompositeDemo.FirstChild transitioned to ALARM at Friday 04
 August, 2023 12:54:46 UTC',
      'reasonData': '{"triggeringAlarms":[{"arn":"arn:aws:cloudwatch:us-
east-1:111122223333:alarm:CompositeDemo.FirstChild","state":
{"value":"ALARM","timestamp":"2023-08-04T12:54:46.138+0000"}}]}',
      'timestamp': '2023-08-04T12:56:46.138+0000'
    },
    'previousState': {
      'value': 'ALARM',
      'reason': 'arn:aws:cloudwatch:us-
east-1:111122223333:alarm:CompositeDemo.FirstChild transitioned to ALARM at Friday 04
 August, 2023 12:54:46 UTC',
      'reasonData': '{"triggeringAlarms":[{"arn":"arn:aws:cloudwatch:us-
east-1:111122223333:alarm:CompositeDemo.FirstChild","state":
{"value":"ALARM","timestamp":"2023-08-04T12:54:46.138+0000"}}]}',
      'timestamp': '2023-08-04T12:54:46.138+0000',
      'actionsSuppressedBy': 'WaitPeriod',
      'actionsSuppressedReason': 'Actions suppressed by WaitPeriod'
    },
    'configuration': {
      'alarmRule': 'ALARM(CompositeDemo.FirstChild) OR
 ALARM(CompositeDemo.SecondChild)',
      'actionsSuppressor': 'CompositeDemo.ActionsSuppressor',
      'actionsSuppressorWaitPeriod': 120,
      'actionsSuppressorExtensionPeriod': 180
    }
  }
}
```

Configuring how CloudWatch alarms treat missing data

Sometimes, not every expected data point for a metric gets reported to CloudWatch. For example, this can happen when a connection is lost, a server goes down, or when a metric reports data only intermittently by design.

CloudWatch enables you to specify how to treat missing data points when evaluating an alarm. This helps you to configure your alarm so that it goes to `ALARM` state only when appropriate for the type of data being monitored. You can avoid false positives when missing data doesn't indicate a problem.

Similar to how each alarm is always in one of three states, each specific data point reported to CloudWatch falls under one of three categories:

- Not breaching (within the threshold)
- Breaching (violating the threshold)
- Missing

For each alarm, you can specify CloudWatch to treat missing data points as any of the following:

- `notBreaching` – Missing data points are treated as "good" and within the threshold
- `breaching` – Missing data points are treated as "bad" and breaching the threshold
- `ignore` – The current alarm state is maintained
- `missing` – If all data points in the alarm evaluation range are missing, the alarm transitions to INSUFFICIENT_DATA.

The best choice depends on the type of metric and the purpose of the alarm. For example, if you are creating an application rollback alarm using a metric that continually reports data, you might want to treat missing data points as breaching, because it might indicate that something is wrong. But for a metric that generates data points only when an error occurs, such as `ThrottledRequests` in Amazon DynamoDB, you would want to treat missing data as `notBreaching`. The default behavior is `missing`.

> **Important**
>
> Alarms configured on Amazon EC2 metrics can temporarily enter the INSUFFICIENT_DATA state if there are missing metric data points. This is rare, but can happen when the metric reporting is interrupted, even when the Amazon EC2 instance is healthy. For alarms on Amazon EC2 metrics that are configured to take stop, terminate, reboot, or recover actions, we recommend that you configure those alarms to treat missing data as `missing`, and to have these alarms trigger only when in the ALARM state.

Choosing the best option for your alarm prevents unnecessary and misleading alarm condition changes, and also more accurately indicates the health of your system.

> **Important**
>
> Alarms that evaluate metrics in the `AWS/DynamoDB` namespace always ignore missing data even if you choose a different option for how the alarm should treat missing data. When an `AWS/DynamoDB` metric has missing data, alarms that evaluate that metric remain in their current state.

How alarm state is evaluated when data is missing

Whenever an alarm evaluates whether to change state, CloudWatch attempts to retrieve a higher number of data points than the number specified as **Evaluation Periods**. The exact number of data points it attempts to retrieve depends on the length of the alarm period and whether it is based on a metric with standard resolution or high resolution. The time frame of the data points that it attempts to retrieve is the *evaluation range*.

Once CloudWatch retrieves these data points, the following happens:

- If no data points in the evaluation range are missing, CloudWatch evaluates the alarm based on the most recent data points collected. The number of data points evaluated is equal to the **Evaluation Periods** for the alarm. The extra data points from farther back in the evaluation range are not needed and are ignored.
- If some data points in the evaluation range are missing, but the total number of existing data points that were successfully retrieved from the evaluation range is equal to or more than the alarm's **Evaluation Periods**, CloudWatch evaluates the alarm state based on the most recent real data points that were successfully retrieved, including the necessary extra data points from farther back in the evaluation range. In this case, the value you set for how to treat missing data is not needed and is ignored.
- If some data points in the evaluation range are missing, and the number of actual data points that were retrieved is lower than the alarm's number of **Evaluation Periods**, CloudWatch fills in the missing data points with the result you specified for how to treat missing data, and then evaluates the alarm. However, all real data points in the evaluation range are included in the evaluation. CloudWatch uses missing data points only as few times as possible.

> **Note**
>
> A particular case of this behavior is that CloudWatch alarms might repeatedly re-evaluate the last set of data points for a period of time after the metric has stopped flowing. This re-evaluation might cause the alarm to change state and re-execute actions, if it had changed state immediately prior to the metric stream stopping. To mitigate this behavior, use shorter periods.

The following tables illustrate examples of the alarm evaluation behavior. In the first table, **Datapoints to Alarm** and **Evaluation Periods** are both 3. CloudWatch retrieves the 5 most recent data points when evaluating the alarm, in case some of the most recent 3 data points are missing. 5 is the evaluation range for the alarm.

Column 1 shows the 5 most recent data points, because the evaluation range is 5. These data points are shown with the most recent data point on the right. 0 is a non-breaching data point, X is a breaching data point, and - is a missing data point.

Column 2 shows how many of the 3 necessary data points are missing. Even though the most recent 5 data points are evaluated, only 3 (the setting for **Evaluation Periods**) are necessary to evaluate the alarm state. The number of data points in Column 2 is the number of data points that must be "filled in", using the setting for how missing data is being treated.

In columns 3-6, the column headers are the possible values for how to treat missing data. The rows in these columns show the alarm state that is set for each of these possible ways to treat missing data.

Data points	# of data points that must be filled	MISSING	IGNORE	BREACHING	NOT BREACHING
0 - X - X	0	`OK`	`OK`	`OK`	`OK`
0 - - - -	2	`OK`	`OK`	`OK`	`OK`
- - - - -	3	`INSUFFICIENT_DATA`	Retain current state	`ALARM`	`OK`

Data points	# of data points that must be filled	MISSING	IGNORE	BREACHING	NOT BREACHING
0 X X - X	0	`ALARM`	`ALARM`	`ALARM`	`ALARM`
- - X - -	2	`ALARM`	Retain current state	`ALARM`	`OK`

In the second row of the preceding table, the alarm stays `OK` even if missing data is treated as breaching, because the one existing data point is not breaching, and this is evaluated along with two missing data points which are treated as breaching. The next time this alarm is evaluated, if the data is still missing it will go to `ALARM`, as that non-breaching data point will no longer be in the evaluation range.

The third row, where all five of the most recent data points are missing, illustrates how the various settings for how to treat missing data affect the alarm state. If missing data points are considered breaching, the alarm goes into ALARM state, while if they are considered not breaching, then the alarm goes into OK state. If missing data points are ignored, the alarm retains the current state it had before the missing data points. And if missing data points are just considered as missing, then the alarm does not have enough recent real data to make an evaluation, and goes into INSUFFICIENT_DATA.

In the fourth row, the alarm goes to `ALARM` state in all cases because the three most recent data points are breaching, and the alarm's **Evaluation Periods** and **Datapoints to Alarm** are both set to 3. In this case, the missing data point is ignored and the setting for how to evaluate missing data is not needed, because there are 3 real data points to evaluate.

Row 5 represents a special case of alarm evaluation called *premature alarm state*. For more information, see [Avoiding premature transitions to alarm state](#).

In the next table, the **Period** is again set to 5 minutes, and **Datapoints to Alarm** is only 2 while **Evaluation Periods** is 3. This is a 2 out of 3, M out of N alarm.

The evaluation range is 5. This is the maximum number of recent data points that are retrieved and can be used in case some data points are missing.

Data points	# of missing data points	MISSING	IGNORE	BREACHING	NOT BREACHING
0 - X - X	0	ALARM	ALARM	ALARM	ALARM
0 0 X 0 X	0	ALARM	ALARM	ALARM	ALARM
0 - X - -	1	OK	OK	ALARM	OK
- - - - 0	2	OK	OK	ALARM	OK
- - - X -	2	ALARM	Retain current state	ALARM	OK

In rows 1 and 2, the alarm always goes to ALARM state because 2 of the 3 most recent data points are breaching. In row 2, the two oldest data points in the evaluation range are not needed because none of the 3 most recent data points are missing, so these two older data points are ignored.

In rows 3 and 4, the alarm goes to ALARM state only if missing data is treated as breaching, in which case the two most recent missing data points are both treated as breaching. In row 4, these two missing data points that are treated as breaching provide the two necessary breaching data points to trigger the ALARM state.

Row 5 represents a special case of alarm evaluation called *premature alarm state*. For more information, see the following section.

Avoiding premature transitions to alarm state

CloudWatch alarm evaluation includes logic to try to avoid false alarms, where the alarm goes into ALARM state prematurely when data is intermittent. The example shown in row 5 in the tables in the previous section illustrate this logic. In those rows, and in the following examples, the **Evaluation Periods** is 3 and the evaluation range is 5 data points. **Datapoints to Alarm** is 3, except for the M out of N example, where **Datapoints to Alarm** is 2.

Suppose an alarm's most recent data is `- - - - X`, with four missing data points and then a breaching data point as the most recent data point. Because the next data point may be non-breaching, the alarm does not go immediately into ALARM state when the data is either `- - - - X` or `- - - X -` and **Datapoints to Alarm** is 3. This way, false positives are avoided when the next data point is non-breaching and causes the data to be `- - - X 0` or `- - X - 0`.

However, if the last few data points are `- - X - -`, the alarm goes into ALARM state even if missing data points are treated as missing. This is because alarms are designed to always go into ALARM state when the oldest available breaching datapoint during the **Evaluation Periods** number of data points is at least as old as the value of **Datapoints to Alarm**, and all other more recent data points are breaching or missing. In this case, the alarm goes into ALARM state even if the total number of datapoints available is lower than M (**Datapoints to Alarm**).

This alarm logic applies to M out of N alarms as well. If the oldest breaching data point during the evaluation range is at least as old as the value of **Datapoints to Alarm**, and all of the more recent data points are either breaching or missing, the alarm goes into ALARM state no matter the value of M (**Datapoints to Alarm**).

High-resolution alarms

If you set an alarm on a high-resolution metric, you can specify a high-resolution alarm with a period of 10 seconds, 20 seconds, or 30 seconds, or you can set a regular alarm with a period of any multiple of 60 seconds. There is a higher charge for high-resolution alarms. For more information about high-resolution metrics, see Publish custom metrics.

Alarms on math expressions

You can set an alarm on the result of a math expression that is based on one or more CloudWatch metrics. A math expression used for an alarm can include as many as 10 metrics. Each metric must be using the same period.

For an alarm based on a math expression, you can specify how you want CloudWatch to treat missing data points. In this case, the data point is considered missing if the math expression doesn't return a value for that data point.

Alarms based on math expressions can't perform Amazon EC2 actions.

For more information about metric math expressions and syntax, see Using math expressions with CloudWatch metrics.

Percentile-based CloudWatch alarms and low data samples

When you set a percentile as the statistic for an alarm, you can specify what to do when there is not enough data for a good statistical assessment. You can choose to have the alarm evaluate the statistic anyway and possibly change the alarm state. Or, you can have the alarm ignore the metric while the sample size is low, and wait to evaluate it until there is enough data to be statistically significant.

For percentiles between 0.5 (inclusive) and 1.00 (exclusive), this setting is used when there are fewer than 10/(1-percentile) data points during the evaluation period. For example, this setting would be used if there were fewer than 1000 samples for an alarm on a p99 percentile. For percentiles between 0 and 0.5 (exclusive), the setting is used when there are fewer than 10/percentile data points.

Common features of CloudWatch alarms

The following features apply to all CloudWatch alarms:

- There is no limit to the number of alarms that you can create. To create or update an alarm, you use the CloudWatch console, the PutMetricAlarm API action, or the put-metric-alarm command in the Amazon CLI.
- Alarm names must contain only UTF-8 characters, and can't contain ASCII control characters
- You can list any or all of the currently configured alarms, and list any alarms in a particular state by using the CloudWatch console, the DescribeAlarms API action, or the describe-alarms command in the Amazon CLI.
- You can disable and enable alarm actions by using the DisableAlarmActions and EnableAlarmActions API actions, or the disable-alarm-actions and enable-alarm-actions commands in the Amazon CLI.
- You can test an alarm by setting it to any state using the SetAlarmState API action or the set-alarm-state command in the Amazon CLI. This temporary state change lasts only until the next alarm comparison occurs.
- You can create an alarm for a custom metric before you've created that custom metric. For the alarm to be valid, you must include all of the dimensions for the custom metric in addition to the metric namespace and metric name in the alarm definition. To do this, you can use the PutMetricAlarm API action, or the put-metric-alarm command in the Amazon CLI.
- You can view an alarm's history using the CloudWatch console, the DescribeAlarmHistory API action, or the describe-alarm-history command in the Amazon CLI. CloudWatch preserves alarm history for 30 days. Each state transition is marked with a unique timestamp. In rare cases, your history might show more than one notification for a state change. The timestamp enables you to confirm unique state changes.

- You can favorite alarms from the *Favorites and recents* option in the CloudWatch console navigation pane by hovering over the alarm that you want to favorite and choosing the star symbol next to it.
- Alarms have an evaluation period quota. The evaluation period is calculated by multiplying the alarm period by the number of evaluation periods used.
  - The maximum evaluation period is seven days for alarms with a period of at least one hour (3600 seconds).
  - The maximum evaluation period is one day for alarms with a shorter period.
  - The maximum evaluation period is one day for alarms that use the custom Lambda data source.

> **Note**
>
> Some Amazon resources don't send metric data to CloudWatch under certain conditions. For example, Amazon EBS might not send metric data for an available volume that is not attached to an Amazon EC2 instance, because there is no metric activity to be monitored for that volume. If you have an alarm set for such a metric, you might notice its state change to `INSUFFICIENT_DATA`. This might indicate that your resource is inactive, and might not necessarily mean that there is a problem. You can specify how each alarm treats missing data. For more information, see Configuring how CloudWatch alarms treat missing data.

Best practice alarm recommendations for Amazon services

CloudWatch provides out-of-the box alarm recommendations. These are CloudWatch alarms that we recommend that you create for metrics that are published by other Amazon services. These recommendations can help you identify the metrics that you should set alarms for to follow best practices for monitoring. The recommendations also suggest the alarm thresholds to set. Following these recommendations can help you not miss important monitoring of your Amazon infrastructure.

To find the alarm recommendations, you use the metrics section of the CloudWatch console, and select the alarm recommendations filter toggle. If you navigate to the recommended alarms in the console and then create a recommended alarm, CloudWatch can pre-fill some of the alarm settings. For some recommended alarms, the alarm threshold value is also pre-filled. You can also use the console to download infrastructure-as-code alarm definitions for recommended alarms, and then use this code to create the alarm in Amazon CloudFormation, the Amazon CLI, or Terraform.

You can also see the list of recommended alarms in Recommended alarms.

You are charged for the alarms that you create, at the same rate as any other alarms that you create in CloudWatch. Using the recommendations incurs no extra charges. For more information, see Amazon CloudWatch Pricing.

Find and create recommended alarms

Follow these steps to find the metrics that CloudWatch recommends that you set alarms for, and optionally to create one of these alarms. The first procedure explains how to find the metrics that have recommended alarms, and how to create one of these alarms.

You can also get a bulk download of infrastructure-as-code alarm definitions for all recommended alarms in an Amazon namespace, such as `AWS/Lambda` or `AWS/S3`. Those instructions are later in this topic.

To find the metrics with recommended alarms, and create a single recommended alarm

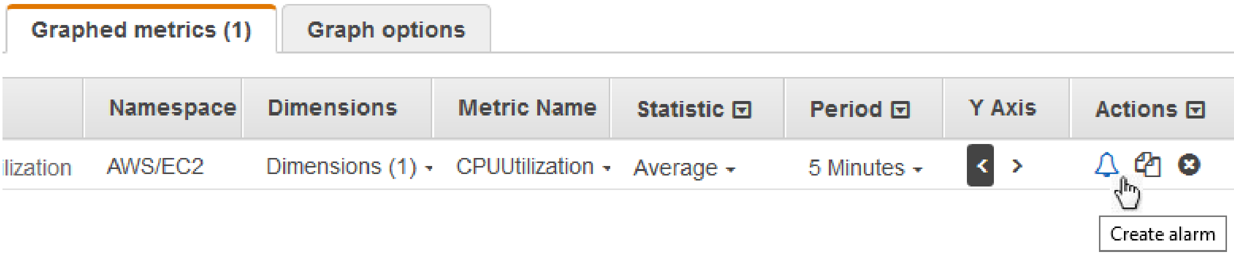
1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Metrics**, **All metrics**.
3. Above the **Metrics** table, Choose **Alarm recommendations**.

   The list of metric namespaces is filtered to include only the metrics that have alarm recommendations and that services in your account are publishing.
4. Choose the namespace for a service.

   The list of metrics under this namespace is filtered to include only those that have alarm recommendations.
5. To see the alarm intent and recommended threshold for a metric, choose **View details**.
6. To create an alarm for one of the metrics, do one of the following:
   - To use the console to create the alarm, do the following:

     a. Select the checkbox for the metric and choose the **Graphed metrics** tab.

     b. Choose the alarm icon.



The alarm creation wizard appears, with the metric name, statistic, and period filled in based on the alarm recommendation. If the recommendation includes a specific threshold value, that value is also pre-filled.

c. Choose **Next**.

d. Under **Notification**, select an SNS topic to notify when the alarm transitions to `ALARM` state, `OK` state, or `INSUFFICIENT_DATA` state.

   To have the alarm send multiple notifications for the same alarm state or for different alarm states, choose **Add notification**.

   To have the alarm not send notifications, choose **Remove**.

e. To have the alarm perform Auto Scaling or EC2 actions, choose the appropriate button and choose the alarm state and action to perform.

f. When finished, choose **Next**.

g. Enter a name and description for the alarm. The name must contain only ASCII characters. Then choose **Next**.

h. Under **Preview and create**, confirm that the information and conditions are what you want, then choose **Create alarm**.

- To download an infrastructure-as-code alarm definition to use in either Amazon CloudFormation, Amazon CLI, or Terraform, choose **Download alarm code** and select the format that you want. The downloaded code will have the recommended settings for the metric name, statistic, and threshold.

To download infrastructure-as-code alarm definitions for all recommended alarms for an Amazon service

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Metrics**, **All metrics**.

3. Above the **Metrics** table, Choose **Alarm recommendations**.

   The list of metric namespaces is filtered to include only the metrics that have alarm recommendations and that services in your account are publishing.

4. Choose the namespace for a service.

   The list of metrics under this namespace is filtered to include only those that have alarm recommendations.

5. The **Download alarm code** displays how many alarms are recommended for the metrics in this namespace. To download infrastructure-as-code alarm definitions for all recommended alarms, choose **Download alarm code** and then choose the code format that you want.

Recommended alarms

The following sections list the metrics that we recommend that you set best practice alarms for. For each metric, the dimensions, alarm intent, recommended threshold, threshold justification, and the period length and number of datapoints is also displayed.

Some metrics might appear twice in the list. This happens when different alarms are recommended for different combinations of dimensions of that metric.

Datapoints to alarm is the number of data points that must be breaching to send the alarm into ALARM state. **Evaluation periods** is the number of periods that are taken into account when the alarm is evaluated. If these numbers are the same, the alarm goes into ALARM state only when that number of consecutive periods have values that breach the threshold. If **Datapoints to alarm** is lower than **Evaluation periods**, then it is an "M out of N" alarm and the alarm goes into ALARM state if at least **Datapoints to alarm** data points are breaching within any **Evaluation periods** set of data points. For more information, see Evaluating an alarm.

Topics

- Amazon EBS
- Amazon EC2
- Amazon ElastiCache
- Amazon ECS
- Amazon ECS with Container Insights
- Amazon ECS with Container Insights with enhanced observability
- Amazon EFS
- Amazon EKS with Container Insights
- Amazon EventBridge Scheduler
- Amazon Kinesis Data Streams
- Lambda
- Lambda Insights
- Amazon VPC (AWS/NATGateway)
- Amazon Private Link (AWS/PrivateLinkEndpoints)
- Amazon Private Link (AWS/PrivateLinkServices)
- Amazon RDS
- Amazon Route 53 Public Data Plane
- Amazon S3
- S3ObjectLambda
- Amazon SNS
- Amazon SQS
- Amazon VPN

Amazon API Gateway

4XXError

Dimensions: ApiName, Stage

Alarm description: This alarm detects a high rate of client-side errors. This can indicate an issue in the authorization or client request parameters. It could also mean that a resource was removed or a client is requesting one that doesn't exist. Consider enabling CloudWatch Logs and checking for any errors that may be causing the 4XX errors. Moreover, consider enabling detailed CloudWatch metrics to view this metric per resource and method and narrow down the source of the errors. Errors could also be caused by exceeding the configured throttling limit. If the responses and logs are reporting high and unexpected rates of 429 errors, follow this guide to troubleshoot this issue.

Intent: This alarm can detect high rates of client-side errors for the API Gateway requests.

Statistic: Average

Recommended threshold: 0.05

Threshold justification: The suggested threshold detects when more than 5% of total requests are getting 4XX errors. However, you can tune the threshold to suit the traffic of the requests as well as acceptable error rates. You can also analyze historical data to determine the acceptable error rate for the application workload and then tune the threshold accordingly. Frequently occurring 4XX errors need to be alarmed on. However, setting a very low value for the threshold can cause the alarm to be too sensitive.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

5XXError

Dimensions: ApiName, Stage

Alarm description: This alarm helps to detect a high rate of server-side errors. This can indicate that there is something wrong on the API backend, the network, or the integration between the API gateway and the backend API. This documentation can help you troubleshoot the cause of 5xx errors.

Intent: This alarm can detect high rates of server-side errors for the API Gateway requests.

Statistic: Average

Recommended threshold: 0.05

Threshold justification: The suggested threshold detects when more than 5% of total requests are getting 5XX errors. However, you can tune the threshold to suit the traffic of the requests as well as acceptable error rates. you can also analyze historical data to determine the acceptable error rate for the application workload and then tune the threshold accordingly. Frequently occurring 5XX errors need to be alarmed on. However, setting a very low value for the threshold can cause the alarm to be too sensitive.

Period: 60

Datapoints to alarm: 3

Evaluation periods: 3

Comparison Operator: GREATER_THAN_THRESHOLD

Count

Dimensions: ApiName, Stage

Alarm description: This alarm helps to detect low traffic volume for the REST API stage. This can be an indicator of an issue with the application calling the API such as using incorrect endpoints. It could also be an indicator of an issue with the configuration or permissions of the API making it unreachable for clients.

Intent: This alarm can detect unexpectedly low traffic volume for the REST API stage. We recommend that you create this alarm if your API receives a predictable and consistent number of requests under normal conditions. If you have detailed CloudWatch metrics enabled and you can predict the normal traffic volume per method and resource, we recommend that you create alternative alarms to have more fine-grained monitoring of traffic volume drops for each resource and method. This alarm is not recommended for APIs that don't expect constant and consistent traffic.

Statistic: SampleCount

Recommended threshold: Depends on your situation

Threshold justification: Set the threshold based on historical data analysis to determine what the expected baseline request count for your API is. Setting the threshold at a very high value might cause the alarm to be too sensitive at periods of normal and expected low traffic. Conversely, setting it at a very low value might cause the alarm to miss anomalous smaller drops in traffic volume.

Period: 60

Datapoints to alarm: 10

Evaluation periods: 10

Comparison Operator: LESS_THAN_THRESHOLD

Count

Dimensions: ApiName, Stage, Resource, Method

Alarm description: This alarm helps to detect low traffic volume for the REST API resource and method in the stage. This can indicate an issue with the application calling the API such as using incorrect endpoints. It could also be an indicator of an issue with the configuration or permissions of the API making it unreachable for clients.

Intent: This alarm can detect unexpectedly low traffic volume for the REST API resource and method in the stage. We recommend that you create this alarm if your API receives a predictable and consistent number of requests under normal conditions. This alarm is not recommended for APIs that don't expect constant and consistent traffic.

Statistic: SampleCount

Recommended threshold: Depends on your situation

Threshold justification: Set the threshold based on historical data analysis to determine what the expected baseline request count for your API is. Setting the threshold at a very high value might cause the alarm to be too sensitive at periods of normal and expected low traffic. Conversely, setting it at a very low value might cause the alarm to miss anomalous smaller drops in traffic volume.

Period: 60

Datapoints to alarm: 10

Evaluation periods: 10

Comparison Operator: LESS_THAN_THRESHOLD

Count

Dimensions: ApiId, Stage

Alarm description: This alarm helps to detect low traffic volume for the HTTP API stage. This can indicate an issue with the application calling the API such as using incorrect endpoints. It could also be an indicator of an issue with the configuration or permissions of the API making it unreachable for clients.

Intent: This alarm can detect unexpectedly low traffic volume for the HTTP API stage. We recommend that you create this alarm if your API receives a predictable and consistent number of requests under normal conditions. If you have detailed CloudWatch metrics enabled and you can predict the normal traffic volume per route, we recommend that you create alternative alarms to this in order to have more fine-grained monitoring of traffic volume drops for each route. This alarm is not recommended for APIs that don't expect constant and consistent traffic.

Statistic: SampleCount

Recommended threshold: Depends on your situation

Threshold justification: Set the threshold value based on historical data analysis to determine what the expected baseline request count for your API is. Setting the threshold at a very high value might cause the alarm to be too sensitive at periods of normal and expected low traffic. Conversely, setting it at a very low value might cause the alarm to miss anomalous smaller drops in traffic volume.

Period: 60

Datapoints to alarm: 10

Evaluation periods: 10

Comparison Operator: LESS_THAN_THRESHOLD

Count

Dimensions: ApiId, Stage, Resource, Method

Alarm description: This alarm helps to detect low traffic volume for the HTTP API route in the stage. This can indicate an issue with the application calling the API such as using incorrect endpoints. It could also indicate an issue with the configuration or permissions of the API making it unreachable for clients.

Intent: This alarm can detect unexpectedly low traffic volume for the HTTP API route in the stage. We recommend that you create this alarm if your API receives a predictable and consistent number of requests under normal conditions. This alarm is not recommended for APIs that don't expect constant and consistent traffic.

Statistic: SampleCount

Recommended threshold: Depends on your situation

Threshold justification: Set the threshold value based on historical data analysis to determine what the expected baseline request count for your API is. Setting the threshold at a very high value might cause the alarm to be too sensitive at periods of normal and expected low traffic. Conversely, setting it at a very low value might cause the alarm to miss anomalous smaller drops in traffic volume.

Period: 60

Datapoints to alarm: 10

Evaluation periods: 10

Comparison Operator: LESS_THAN_THRESHOLD

IntegrationLatency

Dimensions: ApiId, Stage

Alarm description: This alarm helps to detect if there is high integration latency for the API requests in a stage. You can correlate the `IntegrationLatency` metric value with the corresponding latency metric of your backend such as the `Duration` metric for Lambda integrations. This helps you determine whether the API backend is taking more time to process requests from clients due to performance issues, or if there is some other overhead from initialization or cold start. Additionally, consider enabling CloudWatch Logs for your API and checking the logs for any errors that may be causing the high latency issues. Moreover, consider enabling detailed CloudWatch metrics to get a view of this metric per route, to help you narrow down the source of the integration latency.

Intent: This alarm can detect when the API Gateway requests in a stage have a high integration latency. We recommend this alarm for WebSocket APIs, and we consider it optional for HTTP APIs because they already have separate alarm recommendations for the Latency metric. If you have detailed CloudWatch metrics enabled and you have different integration latency performance requirements per route, we recommend that you create alternative alarms in order to have more fine-grained monitoring of the integration latency for each route.

Statistic: p90

Recommended threshold: 2000.0

Threshold justification: The suggested threshold value does not work for all the API workloads. However, you can use it as a starting point for the threshold. You can then choose different threshold values based on the workload and acceptable latency, performance, and SLA requirements for the API. If is acceptable for the API to have a higher latency in general, set a higher threshold value to make the alarm less sensitive. However, if the API is expected to provide near real-time responses, set a lower threshold value. You can also analyze historical data to determine the expected baseline latency for the application workload, and then used to tune the threshold value accordingly.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_OR_EQUAL_TO_THRESHOLD

IntegrationLatency

Dimensions: ApiId, Stage, Route

Alarm description: This alarm helps to detect if there is high integration latency for the WebSocket API requests for a route in a stage. You can correlate the `IntegrationLatency` metric value with the corresponding latency metric of your backend such as the `Duration` metric for Lambda integrations. This helps you determine whether the API backend is taking more time to process requests from clients due to performance issues or if there is some other overhead from initialization or cold start. Additionally, consider enabling CloudWatch Logs for your API and checking the logs for any errors that may be causing the high latency issues.

Intent: This alarm can detect when the API Gateway requests for a route in a stage have high integration latency.

Statistic: p90

Recommended threshold: 2000.0

Threshold justification: The suggested threshold value does not work for all the API workloads. However, you can use it as a starting point for the threshold. You can then choose different threshold values based on the workload and acceptable latency, performance, and SLA requirements for the API. If it is acceptable for the API to have a higher latency in general, you can set a higher threshold value to make the alarm less sensitive. However, if the API is expected to provide near real-time responses, set a lower threshold value. You can also analyze historical data to determine the expected baseline latency for the application workload, and then used to tune the threshold value accordingly.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_OR_EQUAL_TO_THRESHOLD

Latency

Dimensions: ApiName, Stage

Alarm description: This alarm detects high latency in a stage. Find the `IntegrationLatency` metric value to check the API backend latency. If the two metrics are mostly aligned, the API backend is the source of higher latency and you should investigate there for issues. Consider also enabling CloudWatch Logs and checking for errors that might be causing the high latency. Moreover, consider enabling detailed CloudWatch metrics to view this metric per resource and method and narrow down the source of the latency. If applicable, refer to the troubleshooting with Lambda or troubleshooting for edge-optimized API endpoints guides.

Intent: This alarm can detect when the API Gateway requests in a stage have high latency. If you have detailed CloudWatch metrics enabled and you have different latency performance requirements for each method and resource, we recommend that you create alternative alarms to have more fine-grained monitoring of the latency for each resource and method.

Statistic: p90

Recommended threshold: 2500.0

Threshold justification: The suggested threshold value does not work for all API workloads. However, you can use it as a starting point for the threshold. You can then choose different threshold values based on the workload and acceptable latency, performance, and SLA requirements for the API. If it is acceptable for the API to have a higher latency in general, you can set a higher threshold value to make the alarm less sensitive. However, if the API is expected to provide near real-time responses, set a lower threshold value. You can also analyze historical data to determine what the expected baseline latency is for the application workload and then tune the threshold value accordingly.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_OR_EQUAL_TO_THRESHOLD

Latency

Dimensions: ApiName, Stage, Resource, Method

Alarm description: This alarm detects high latency for a resource and method in a stage. Find the `IntegrationLatency` metric value to check the API backend latency. If the two metrics are mostly aligned, the API backend is the source of higher latency and you should investigate there for performance issues. Consider also enabling CloudWatch Logs and checking for any errors that might be causing the high latency. You can also refer to the troubleshooting with Lambda or troubleshooting for edge-optimized API endpoints guides if applicable.

Intent: This alarm can detect when the API Gateway requests for a resource and method in a stage have high latency.

Statistic: p90

Recommended threshold: 2500.0

Threshold justification: The suggested threshold value does not work for all the API workloads. However, you can use it as a starting point for the threshold. You can then choose different threshold values based on the workload and acceptable latency, performance, and SLA requirements for the API. If it is acceptable for the API to have a higher latency in general, you can set a higher threshold value to make the alarm less sensitive. However, if the API is expected to provide near real-time responses, set a lower threshold value. You can also analyze historical data to determine the expected baseline latency for the application workload and then tune the threshold value accordingly.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_OR_EQUAL_TO_THRESHOLD

Latency

Dimensions: ApiId, Stage

Alarm description: This alarm detects high latency in a stage. Find the `IntegrationLatency` metric value to check the API backend latency. If the two metrics are mostly aligned, the API backend is the source of higher latency and you should investigate there for performance issues. Consider also enabling CloudWatch Logs and checking for any errors that may be causing the high latency. Moreover, consider enabling detailed CloudWatch metrics to view this metric per route and narrow down the source of the latency. You can also refer to the troubleshooting with Lambda integrations guide if applicable.

Intent: This alarm can detect when the API Gateway requests in a stage have high latency. If you have detailed CloudWatch metrics enabled and you have different latency performance requirements per route, we recommend that you create alternative alarms to have more fine-grained monitoring of the latency for each route.

Statistic: p90

Recommended threshold: 2500.0

Threshold justification: The suggested threshold value does not work for all the API workloads. However, it can be used as a starting point for the threshold. You can then choose different threshold values based on the workload and acceptable latency, performance and SLA requirements for the API. If it is acceptable for the API to have a higher latency in general, you can set a higher threshold value to make it less sensitive.However, if the API is expected to provide near real-time responses, set a lower threshold value. You can also analyze historical data to determine the expected baseline latency for the application workload and then tune the threshold value accordingly.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_OR_EQUAL_TO_THRESHOLD

Latency

Dimensions: ApiId, Stage, Resource, Method

Alarm description: This alarm detects high latency for a route in a stage. Find the `IntegrationLatency` metric value to check the API backend latency. If the two metrics are mostly aligned, the API backend is the source of higher latency and should be investigated for performance issues. Consider also enabling CloudWatch logs and checking for any errors that might be causing the high latency. You can also refer to the troubleshooting with Lambda integrations guide if applicable.

Intent: This alarm is used to detect when the API Gateway requests for a route in a stage have high latency.

Statistic: p90

Recommended threshold: 2500.0

Threshold justification: The suggested threshold value does not work for all the API workloads. However, it can be used as a starting point for the threshold. You can then choose different threshold values based on the workload and acceptable latency, performance, and SLA requirements for the API. If it is acceptable for the API to have a higher latency in general, you can set a higher threshold value to make the alarm less sensitive. However, if the API is expected to provide near real-time responses, set a lower threshold value. You can also analyze historical data to determine the expected baseline latency for the application workload and then tune the threshold value accordingly.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_OR_EQUAL_TO_THRESHOLD

4xx

Dimensions: ApiId, Stage

Alarm description: This alarm detects a high rate of client-side errors. This can indicate an issue in the authorization or client request parameters. It could also mean that a route was removed or a client is requesting one that doesn't exist in the API. Consider enabling CloudWatch Logs and checking for any errors that may be causing the 4xx errors. Moreover, consider enabling detailed CloudWatch metrics to view this metric per route, to help you narrow down the source of the errors. Errors can also be caused by exceeding the configured throttling limit. If the responses and logs are reporting high and unexpected rates of 429 errors, follow this guide to troubleshoot this issue.

Intent: This alarm can detect high rates of client-side errors for the API Gateway requests.

Statistic: Average

Recommended threshold: 0.05

Threshold justification: The suggested threshold detects when more than 5% of total requests are getting 4xx errors. However, you can tune the threshold to suit the traffic of the requests as well as acceptable error rates. You can also analyze historical data to determine the acceptable error rate for the application workload and then tune the threshold accordingly. Frequently occurring 4xx errors need to be alarmed on. However, setting a very low value for the threshold can cause the alarm to be too sensitive.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

5xx

Dimensions: ApiId, Stage

Alarm description: This alarm helps to detect a high rate of server-side errors. This can indicate that there is something wrong on the API backend, the network, or the integration between the API gateway and the backend API. This documentation can help you troubleshoot the cause for 5xx errors.

Intent: This alarm can detect high rates of server-side errors for the API Gateway requests.

Statistic: Average

Recommended threshold: 0.05

Threshold justification: The suggested threshold detects when more than 5% of total requests are getting 5xx errors. However, you can tune the threshold to suit the traffic of the requests as well as acceptable error rates. You can also analyze historical data to determine what the acceptable error rate is for the application workload, and then you can tune the threshold accordingly. Frequently occurring 5xx errors need to be alarmed on. However, setting a very low value for the threshold can cause the alarm to be too sensitive.

Period: 60

Datapoints to alarm: 3

Evaluation periods: 3

Comparison Operator: GREATER_THAN_THRESHOLD

MessageCount

Dimensions: ApiId, Stage

Alarm description: This alarm helps to detect low traffic volume for the WebSocket API stage. This can indicate an issue when clients call the API such as using incorrect endpoints, or issues with the backend sending messages to clients. It could also indicate an issue with the configuration or permissions of the API, making it unreachable for clients.

Intent: This alarm can detect unexpectedly low traffic volume for the WebSocket API stage. We recommend that you create this alarm if your API receives and sends a predictable and consistent number of messages under normal conditions. If you have detailed CloudWatch metrics enabled and you can predict the normal traffic volume per route, it is better to create alternative alarms to this one, in order to have more fine-grained monitoring of traffic volume drops for each route. We do not recommend this alarm for APIs that don't expect constant and consistent traffic.

Statistic: SampleCount

Recommended threshold: Depends on your situation

Threshold justification: Set the threshold value based on historical data analysis to determine what the expected baseline message count for your API is. Setting the threshold to a very high value might cause the alarm to be too sensitive at periods of normal and expected low traffic. Conversely, setting it to a very low value might cause the alarm to miss anomalous smaller drops in traffic volume.

Period: 60

Datapoints to alarm: 10

Evaluation periods: 10

Comparison Operator: LESS_THAN_THRESHOLD

MessageCount

Dimensions: ApiId, Stage, Route

Alarm description: This alarm helps detect low traffic volume for the WebSocket API route in the stage. This can indicate an issue with the clients calling the API such as using incorrect endpoints, or issues with the backend sending messages to clients. It could also indicate an issue with the configuration or permissions of the API, making it unreachable for clients.

Intent: This alarm can detect unexpectedly low traffic volume for the WebSocket API route in the stage. We recommend that you create this alarm if your API receives and sends a predictable and consistent number of messages under normal conditions. We do not recommend this alarm for APIs that don't expect constant and consistent traffic.

Statistic: SampleCount

Recommended threshold: Depends on your situation

Threshold justification: Set the threshold based on historical data analysis to determine what the expected baseline message count for your API is. Setting the threshold to a very high value might cause the alarm to be too sensitive at periods of normal and expected low traffic. Conversely, setting it to a very low value might cause the alarm to miss anomalous smaller drops in traffic volume.

Period: 60

Datapoints to alarm: 10

Evaluation periods: 10

Comparison Operator: LESS_THAN_THRESHOLD

ClientError

Dimensions: ApiId, Stage

Alarm description: This alarm detects a high rate of client errors. This can indicate an issue in the authorization or message parameters. It could also mean that a route was removed or a client is requesting one that doesn't exist in the API. Consider enabling CloudWatch Logs and checking for any errors that may be causing the 4xx errors. Moreover, consider enabling detailed CloudWatch metrics to view this metric per route, to help you narrow down the source of the errors. Errors could also be caused by exceeding the configured throttling limit. If the responses and logs are reporting high and unexpected rates of 429 errors, follow this guide to troubleshoot this issue.

Intent: This alarm can detect high rates of client errors for the WebSocket API Gateway messages.

Statistic: Average

Recommended threshold: 0.05

Threshold justification: The suggested threshold detects when more than 5% of total requests are getting 4xx errors. You can tune the threshold to suit the traffic of the requests as well as to suit your acceptable error rates. You can also analyze historical data to determine the acceptable error rate for the application workload, and then tune the threshold accordingly. Frequently occurring 4xx errors need to be alarmed on. However, setting a very low value for the threshold can cause the alarm to be too sensitive.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

ExecutionError

Dimensions: ApiId, Stage

Alarm description: This alarm helps to detect a high rate of execution errors. This can be caused by 5xx errors from your integration, permission issues, or other factors preventing successful invocation of the integration, such as the integration being throttled or deleted. Consider enabling CloudWatch Logs for your API and checking the logs for the type and cause of the errors. Moreover, consider enabling detailed CloudWatch metrics to get a view of this metric per route, to help you narrow down the source of the errors. This documentation can also help you troubleshoot the cause of any connection errors.

Intent: This alarm can detect high rates of execution errors for the WebSocket API Gateway messages.

Statistic: Average

Recommended threshold: 0.05

Threshold justification: The suggested threshold detects when more than 5% of total requests are getting execution errors. You can tune the threshold to suit the traffic of the requests, as well as to suit your acceptable error rates. You can analyze historical data to determine the acceptable error rate for the application workload, and then tune the threshold accordingly. Frequently occurring execution errors need to be alarmed on. However, setting a very low value for the threshold can cause the alarm to be too sensitive.

Period: 60

Datapoints to alarm: 3

Evaluation periods: 3

Comparison Operator: GREATER_THAN_THRESHOLD

Amazon EC2 Auto Scaling

GroupInServiceCapacity

Dimensions: AutoScalingGroupName

Alarm description: This alarm helps to detect when the capacity in the group is below the desired capacity required for your workload. To troubleshoot, check your scaling activities for launch failures and confirm that your desired capacity configuration is correct.

Intent: This alarm can detect a low availability in your auto scaling group because of launch failures or suspended launches.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: The threshold value should be the minimum capacity required to run your workload. In most cases, you can set this to match the GroupDesiredCapacity metric.

Period: 60

Datapoints to alarm: 10

Evaluation periods: 10

Comparison Operator: LESS_THAN_THRESHOLD

Amazon Certificate Manager (ACM)

DaysToExpiry

Dimensions: CertificateArn

Alarm description: This alarm helps you detect when a certificate managed by or imported into ACM is approaching its expiration date. It helps to prevent unexpected certificate expirations that could lead to service disruptions. When the alarm transitions into ALARM state, you should take immediate action to renew or re-import the certificate. For ACM-managed certificates, see the instructions at certificate renewal process. For imported certificates, see the instructions at re-import process.

Intent: This alarm can proactively alert you about upcoming certificate expirations. It provides sufficient advance notice to allow for manual intervention, enabling you to renew or replace certificates before they expire. This helps you maintain the security and availability of TLS-enabled services. When this goes into ALARM, immediately investigate the certificate status and initiate the renewal process if necessary.

Statistic: Minimum

Recommended threshold: 44.0

Threshold justification: The 44-day threshold provides a balance between early warning and avoiding false alarms. It allows sufficient time for manual intervention if automatic renewal fails. Adjust this value based on your certificate renewal process and operational response times.

Period: 86400

Datapoints to alarm: 1

Evaluation periods: 1

Comparison Operator: LESS_THAN_OR_EQUAL_TO_THRESHOLD

Amazon CloudFront

5xxErrorRate

Dimensions: DistributionId, Region=Global

Alarm description: This alarm monitors the percentage of 5xx error responses from your origin server, to help you detect if the CloudFront service is having issues. See Troubleshooting error responses from your origin for information to help you understand the problems with your server. Also, turn on additional metrics to get detailed error metrics.

Intent: This alarm is used to detect problems with serving requests from the origin server, or problems with communication between CloudFront and your origin server.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: The recommended threshold value for this alarm is highly dependent on the tolerance for 5xx responses. You can analyze historical data and trends, and then set the threshold accordingly. Because 5xx errors can be caused by transient issues, we recommend that you set the threshold to a value greater than 0 so that the alarm is not too sensitive.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

OriginLatency

Dimensions: DistributionId, Region=Global

Alarm description: The alarm helps to monitor if the origin server is taking too long to respond. If the server takes too long to respond, it might lead to a timeout. Refer to find and fix delayed responses from applications on your origin server if you experience consistently high `OriginLatency` values.

Intent: This alarm is used to detect problems with the origin server taking too long to respond.

Statistic: p90

Recommended threshold: Depends on your situation

Threshold justification: You should calculate the value of about 80% of the origin response timeout, and use the result as the threshold value. If this metric is consistently close to the origin response timeout value, you might start experiencing 504 errors.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

FunctionValidationErrors

Dimensions: DistributionId, FunctionName, Region=Global

Alarm description: This alarm helps you monitor validation errors from CloudFront functions so that you can take steps to resolve them. Analyze the CloudWatch function logs and look at the function code to find and resolve the root cause of the problem. See Restrictions on edge functions to understand the common misconfigurations for CloudFront Functions.

Intent: This alarm is used to detect validation errors from CloudFront functions.

Statistic: Sum

Recommended threshold: 0.0

Threshold justification: A value greater than 0 indicates a validation error. We recommend setting the threshold to 0 because validation errors imply a problem when CloudFront functions hand off back to CloudFront. For example, CloudFront needs the HTTP Host header in order to process a request. There is nothing stopping a user from deleting the Host header in their CloudFront functions code. But when CloudFront gets the response back and the Host header is missing, CloudFront throws a validation error.

Period: 60

Datapoints to alarm: 2

Evaluation periods: 2

Comparison Operator: GREATER_THAN_THRESHOLD

FunctionExecutionErrors

Dimensions: DistributionId, FunctionName, Region=Global

Alarm description: This alarm helps you monitor execution errors from CloudFront functions so that you can take steps to resolve them. Analyze the CloudWatch function logs and look at the function code to find and resolve the root cause of the problem.

Intent: This alarm is used to detect execution errors from CloudFront functions.

Statistic: Sum

Recommended threshold: 0.0

Threshold justification: We recommend to set the threshold to 0 because an execution error indicates a problem with the code that occurs at runtime.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

FunctionThrottles

Dimensions: DistributionId, FunctionName, Region=Global

Alarm description: This alarm helps you to monitor if your CloudFront function is throttled. If your function is throttled, it means that it is taking too long to execute. To avoid function throttles, consider optimizing the function code.

Intent: This alarm can detect when your CloudFront function is throttled so that you can react and resolve the issue for a smooth customer experience.

Statistic: Sum

Recommended threshold: 0.0

Threshold justification: We recommend setting the threshold to 0, to allow quicker resolution of the function throttles.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

Amazon Cognito

SignUpThrottles

Dimensions: UserPool, UserPoolClient

Alarm description: This alarm monitors the count of throttled requests. If users are consistently getting throttled, you should increase the limit by requesting a service quota increase. Refer to Quotas in Amazon Cognito to learn how to request a quota increase. To take actions proactively, consider tracking the usage quota.

Intent: This alarm helps to monitor the occurrence of throttled sign-up requests. This can help you know when to take actions to mitigate any degradation in sign-up experience. Sustained throttling of requests is a negative user sign-up experience.

Statistic: Sum

Recommended threshold: Depends on your situation

Threshold justification: A well-provisioned user pool should not encounter any throttling which spans across multiple data points. So, a typical threshold for an expected workload should be zero. For an irregular workload with frequent bursts, you can analyze historical data to determine the acceptable throttling for the application workload, and then you can tune the threshold accordingly. A throttled request should be retried to minimize the impact on the application.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

SignInThrottles

Dimensions: UserPool, UserPoolClient

Alarm description: This alarm monitors the count of throttled user authentication requests. If users are consistently getting throttled, you might need to increase the limit by requesting a service quota increase. Refer to Quotas in Amazon Cognito to learn how to request a quota increase. To take actions proactively, consider tracking the usage quota.

Intent: This alarm helps to monitor the occurrence of throttled sign-in requests. This can help you know when to take actions to mitigate any degradation in sign-in experience. Sustained throttling of requests is a bad user authentication experience.

Statistic: Sum

Recommended threshold: Depends on your situation

Threshold justification: A well-provisioned user pool should not encounter any throttling which spans across multiple data points. So, a typical threshold for an expected workload should be zero. For an irregular workload with frequent bursts, you can analyze historical data to determine the acceptable throttling for the application workload, and then you can tune the threshold accordingly. A throttled request should be retried to minimize the impact on the application.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

TokenRefreshThrottles

Dimensions: UserPool, UserPoolClient

Alarm description: You can set the threshold value can to suit the traffic of the request as well as to match acceptable throttling for token refresh requests. Throttling is used to protect your system from too many requests. However, it is important to monitor if you are under provisioned for your normal traffic as well. You can analyze historical data to find the acceptable throttling for the application workload, and then you can tune your alarm threshold to be higher than your acceptable throttling level. Throttled requests should be retried by the application/service as they are transient. Therefore, a very low value for the threshold can cause alarm to be sensitive.

Intent: This alarm helps to monitor the occurrence of throttled token refresh requests. This can help you know when to take actions to mitigate any potential problems, to ensure a smooth user experience and the health and reliability of your authentication system. Sustained throttling of requests is a bad user authentication experience.

Statistic: Sum

Recommended threshold: Depends on your situation

Threshold justification: Threshold value can also be set/tuned to suit the traffic of the request as well as acceptable throttling for token refresh requests. Throttling are there for protecting your system from too many requests, however it is important to monitor if you are under provisioned for your normal traffic as well and see if it is causing the impact. Historical data can also be analyzed to see what is the acceptable throttling for the application workload and threshold can be tuned higher than your usual acceptable throttling level. Throttled requests should be retried by the application/service as they are transient. Therefore, a very low value for the threshold can cause alarm to be sensitive.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

FederationThrottles

Dimensions: UserPool, UserPoolClient, IdentityProvider

Alarm description: This alarm monitors the count of throttled identity federation requests. If you consistently see throttling, it might indicate that you need to increase the limit by requesting a service quota increase. Refer to Quotas in Amazon Cognito to learn how to request a quota increase.

Intent: This alarm helps to monitor the occurrence of throttled identity federation requests. This can help you take proactive responses to performance bottlenecks or misconfigurations, and ensure a smooth authentication experience for your users. Sustained throttling of requests is a bad user authentication experience.

Statistic: Sum

Recommended threshold: Depends on your situation

Threshold justification: You can set the threshold to suit the traffic of the request as well as to match the acceptable throttling for identity federation requests. Throttling is used for protecting your system from too many requests. However, it is important to monitor if you are under provisioned for your normal traffic as well. You can analyze historical data to find the acceptable throttling for the application workload, and then set the threshold to a value above your acceptable throttling level. Throttled requests should be retried by the application/ service as they are transient. Therefore, a very low value for the threshold can cause alarm to be sensitive.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

Amazon DynamoDB

AccountProvisionedReadCapacityUtilization

Dimensions: None

Alarm description: This alarm detects if the account's read capacity is reaching its provisioned limit. You can raise the account quota for read capacity utilization if this occurs. You can view your current quotas for read capacity units and request increases using Service Quotas.

Intent: The alarm can detect if the account's read capacity utilization is approaching its provisioned read capacity utilization. If the utilization reaches its maximum limit, DynamoDB starts to throttle read requests.

Statistic: Maximum

Recommended threshold: 80.0

Threshold justification: Set the threshold to 80%, so that action (such as raising the account limits) can be taken before it reaches full capacity to avoid throttling.

Period: 300

Datapoints to alarm: 2

Evaluation periods: 2

Comparison Operator: GREATER_THAN_THRESHOLD

AccountProvisionedWriteCapacityUtilization

Dimensions: None

Alarm description: This alarm detects if the account's write capacity is reaching its provisioned limit. You can raise the account quota for write capacity utilization if this occurs. You can view your current quotas for write capacity units and request increases using Service Quotas.

Intent: This alarm can detect if the account's write capacity utilization is approaching its provisioned write capacity utilization. If the utilization reaches its maximum limit, DynamoDB starts to throttle write requests.

Statistic: Maximum

Recommended threshold: 80.0

Threshold justification: Set the threshold to 80%, so that the action (such as raising the account limits) can be taken before it reaches full capacity to avoid throttling.

Period: 300

Datapoints to alarm: 2

Evaluation periods: 2

Comparison Operator: GREATER_THAN_THRESHOLD

AgeOfOldestUnreplicatedRecord

Dimensions: TableName, DelegatedOperation

Alarm description: This alarm detects the delay in replication to a Kinesis data stream. Under normal operation, `AgeOfOldestUnreplicatedRecord` should be only milliseconds. This number grows based on unsuccessful replication attempts caused by customer-controlled configuration choices. Customer-controlled configuration examples that lead to unsuccessful replication attempts are an under-provisioned Kinesis data stream capacity that leads to excessive throttling. or a manual update to the Kinesis data stream's access policies that prevents DynamoDB from adding data to the data stream. To keep this metric as low as possible, you need to ensure the right provisioning of Kinesis data stream capacity and make sure that DynamoDB's permissions are unchanged.

Intent: This alarm can monitor unsuccessful replication attempts and the resulting delay in replication to the Kinesis data stream.

Statistic: Maximum

Recommended threshold: Depends on your situation

Threshold justification: Set the threshold according to the desired replication delay measured in milliseconds. This value depends on your workload's requirements and expected performance.

Period: 300

Datapoints to alarm: 3

Evaluation periods: 3

Comparison Operator: GREATER_THAN_THRESHOLD

FailedToReplicateRecordCount

Dimensions: TableName, DelegatedOperation

Alarm description: This alarm detects the number of records that DynamoDB failed to replicate to your Kinesis data stream. Certain items larger than 34 KB might expand in size to change data records that are larger than the 1 MB item size limit of Kinesis Data Streams. This size expansion occurs when these larger than 34 KB items include a large number of Boolean or empty attribute values. Boolean and empty attribute values are stored as 1 byte in DynamoDB, but expand up to 5 bytes when they're serialized using standard JSON for Kinesis Data Streams replication. DynamoDB can't replicate such change records to your Kinesis data stream. DynamoDB skips these change data records, and automatically continues replicating subsequent records.

Intent: This alarm can monitor the number of records that DynamoDB failed to replicate to your Kinesis data stream because of the item size limit of Kinesis Data Streams.

Statistic: Sum

Recommended threshold: 0.0

Threshold justification: Set the threshold to 0 to detect any records that DynamoDB failed to replicate.

Period: 60

Datapoints to alarm: 1

Evaluation periods: 1

Comparison Operator: GREATER_THAN_THRESHOLD

ReadThrottleEvents

Dimensions: TableName

Alarm description: This alarm detects if there are high number of read requests getting throttled for the DynamoDB table. To troubleshoot the issue, see Troubleshooting throttling issues in Amazon DynamoDB.

Intent: This alarm can detect sustained throttling for read requests to the DynamoDB table. Sustained throttling of read requests can negatively impact your workload read operations and reduce the overall efficiency of the system.

Statistic: Sum

Recommended threshold: Depends on your situation

Threshold justification: Set the threshold according to the expected read traffic for the DynamoDB table, accounting for an acceptable level of throttling. It is important to monitor whether you are under provisioned and not causing consistent throttling. You can also analyze historical data to find the acceptable throttling level for the application workload, and then tune the threshold to be higher than your usual throttling level. Throttled requests should be retried by the application or service as they are transient. Therefore, a very low threshold may cause the alarm to be too sensitive, causing unwanted state transitions.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

ReadThrottleEvents

Dimensions: TableName, GlobalSecondaryIndexName

Alarm description: This alarm detects if there are a high number of read requests getting throttled for the Global Secondary Index of the DynamoDB table. To troubleshoot the issue, see Troubleshooting throttling issues in Amazon DynamoDB.

Intent: The alarm can detect sustained throttling for read requests for the Global Secondary Index of the DynamoDB Table. Sustained throttling of read requests can negatively impact your workload read operations and reduce the overall efficiency of the system.

Statistic: Sum

Recommended threshold: Depends on your situation

Threshold justification: Set the threshold according to the expected read traffic for the DynamoDB table, accounting for an acceptable level of throttling. It is important to monitor if you are under provisioned and not causing consistent throttling. You can also analyze historical data to find an acceptable throttling level for the application workload, and then tune the threshold to be higher than your usual acceptable throttling level. Throttled requests should be retried by the application or service as they are transient. Therefore, a very low threshold may cause the alarm to be too sensitive, causing unwanted state transitions.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

ReplicationLatency

Dimensions: TableName, ReceivingRegion

Alarm description: The alarm detects if the replica in a Region for the global table is lagging behind the source Region. The latency can increase if an Amazon Region becomes degraded and you have a replica table in that Region. In this case, you can temporarily redirect your application's read and write activity to a different Amazon Region. If you are using 2017.11.29 (Legacy) of global tables, you should verify that write capacity units (WCUs) are identical for each of the replica tables. You can also make sure to follow recommendations in Best practices and requirements for managing capacity.

Intent: The alarm can detect if the replica table in a Region is falling behind replicating the changes from another Region. This could cause your replica to diverge from the other replicas. It's useful to know the replication latency of each Amazon Region and alert if that replication latency increases continually. The replication of the table applies to global tables only.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: The recommended threshold value for this alarm is highly dependent on your use case. Replication latencies longer than 3 minutes are generally a cause for investigation. Review the criticality and requirements of replication delay and analyze historical trends, and then select the threshold accordingly.

Period: 60

Datapoints to alarm: 15

Evaluation periods: 15

Comparison Operator: GREATER_THAN_THRESHOLD

SuccessfulRequestLatency

Dimensions: TableName, Operation

Alarm description: This alarm detects a high latency for the DynamoDB table operation (indicated by the dimension value of the `Operation` in the alarm). See this troubleshooting document for troubleshooting latency issues in Amazon DynamoDB.

Intent: This alarm can detect a high latency for the DynamoDB table operation. Higher latency for the operations can negatively impact the overall efficiency of the system.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: DynamoDB provides single-digit millisecond latency on average for singleton operations such as GetItem, PutItem, and so on. However, you can set the threshold based on acceptable tolerance for the latency for the type of operation and table involved in the workload. You can analyze historical data of this metric to find the usual latency for the table operation, and then set the threshold to a number which represents critical delay for the operation.

Period: 60

Datapoints to alarm: 10

Evaluation periods: 10

Comparison Operator: GREATER_THAN_THRESHOLD

SystemErrors

Dimensions: TableName

Alarm description: This alarm detects a sustained high number of system errors for the DynamoDB table requests. If you continue to get 5xx errors, open the Amazon Service Health Dashboard to check for operational issues with the service. You can use this alarm to get notified in case there is a prolonged internal service issue from DynamoDB and it helps you correlate with the issue your client application is facing. Refer Error handling for DynamoDB for more information.

Intent: This alarm can detect sustained system errors for the DynamoDB table requests. System errors indicate internal service errors from DynamoDB and helps correlate to the issue that the client is having.

Statistic: Sum

Recommended threshold: Depends on your situation

Threshold justification: Set the threshold according to the expected traffic, accounting for an acceptable level of system errors. You can also analyze historical data to find the acceptable error count for the application workload, and then tune the threshold accordingly. System errors should be retried by the application/service as they are transient. Therefore, a very low threshold might cause the alarm to be too sensitive, causing unwanted state transitions.

Period: 60

Datapoints to alarm: 15

Evaluation periods: 15

Comparison Operator: GREATER_THAN_THRESHOLD

ThrottledPutRecordCount

Dimensions: TableName, DelegatedOperation

Alarm description: This alarm detects the records getting throttled by your Kinesis data stream during the replication of change data capture to Kinesis. This throttling happens because of insufficient Kinesis data stream capacity. If you experience excessive and regular throttling, you might need to increase the number of Kinesis stream shards proportionally to the observed write throughput of your table. To learn more about determining the size of a Kinesis data stream, see Determining the Initial Size of a Kinesis Data Stream.

Intent: This alarm can monitor the number of records that that were throttled by your Kinesis data stream because of insufficient Kinesis data stream capacity.

Statistic: Maximum

Recommended threshold: Depends on your situation

Threshold justification: You might experience some throttling during exceptional usage peaks, but throttled records should remain as low as possible to avoid higher replication latency (DynamoDB retries sending throttled records to the Kinesis data stream). Set the threshold to a number which can help you catch regular excessive throttling. You can also analyze historical data of this metric to find the acceptable throttling rates for the application workload. Tune the threshold to a value that the application can tolerate based on your use case.

Period: 60

Datapoints to alarm: 10

Evaluation periods: 10

Comparison Operator: GREATER_THAN_THRESHOLD

UserErrors

Dimensions: None

Alarm description: This alarm detects a sustained high number of user errors for the DynamoDB table requests. You can check client application logs during the issue time frame to see why the requests are invalid. You can check HTTP status code 400 to see the type of error you are getting and take action accordingly. You might have to fix the application logic to create valid requests.

Intent: This alarm can detect sustained user errors for the DynamoDB table requests. User errors for requested operations mean that the client is producing invalid requests and it is failing.

Statistic: Sum

Recommended threshold: Depends on your situation

Threshold justification: Set the threshold to zero to detect any client side errors. Or you can set it to a higher value if you want to avoid the alarm triggering for a very lower number of errors. Decide based on your use case and traffic for the requests.

Period: 60

Datapoints to alarm: 10

Evaluation periods: 10

Comparison Operator: GREATER_THAN_THRESHOLD

WriteThrottleEvents

Dimensions: TableName

Alarm description: This alarm detects if there are a high number of write requests getting throttled for the DynamoDB table. See [Troubleshooting throttling issues in Amazon DynamoDB] to troubleshoot the issue.

Intent: This alarm can detect sustained throttling for write requests to the DynamoDB table. Sustained throttling of write requests can negatively impact your workload write operations and reduce the overall efficiency of the system.

Statistic: Sum

Recommended threshold: Depends on your situation

Threshold justification: Set the threshold according to the expected write traffic for the DynamoDB table, accounting for an acceptable level of throttling. It is important to monitor if you are under provisioned and not causing consistent throttling. You can also analyze historical data to find the acceptable level of throttling for the application workload, and then tune the threshold to a value higher than your usual acceptable throttling level. Throttled requests should be retried by the application/service as they are transient. Therefore, a very low threshold might cause the alarm to be too sensitive, causing unwanted state transitions.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

WriteThrottleEvents

Dimensions: TableName, GlobalSecondaryIndexName

Alarm description: This alarm detects if there are a high number of write requests getting throttled for Global Secondary Index of the DynamoDB table. See [Troubleshooting throttling issues in Amazon DynamoDB]() to troubleshoot the issue.

Intent: This alarm can detect sustained throttling for write requests for the Global Secondary Index of DynamoDB Table. Sustained throttling of write requests can negatively impact your workload write operations and reduce the overall efficiency of the system.

Statistic: Sum

Recommended threshold: Depends on your situation

Threshold justification: Set the threshold according to the expected Write traffic for the DynamoDB table, accounting for an acceptable level of throttling. It is important to monitor if you are under provisioned and not causing consistent throttling. You can also analyze historical data to find the acceptable throttling level for the application workload, and then tune the threshold to a value higher than your usual acceptable throttling level. Throttled requests should be retried by the application/service as they are transient. Therefore, a very low value might cause the alarm to be too sensitive, causing unwanted state transitions.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

Amazon EBS

VolumeStalledIOCheck

Dimensions: VolumeId, InstanceId

Alarm description: This alarm helps you monitor the IO performance of your Amazon EBS volumes. This check detects underlying issues with the Amazon EBS infrastructure, such as hardware or software issues on the storage subsystems underlying the Amazon EBS volumes, hardware issues on the physical host that impact the reachability of the Amazon EBS volumes from your Amazon EC2 instance, and can detect connectivity issues between the instance and the Amazon EBS volumes. If the Stalled IO Check fails, you can either wait for Amazon to resolve the issue, or you can take action such as replacing the affected volume or stopping and restarting the instance to which the volume is attached. In most cases, when this metric fails, Amazon EBS will automatically diagnose and recover your volume within a few minutes.

Intent: This alarm can detect the status of your Amazon EBS volumes to determine when these volumes are impaired and can not complete I/O operations.

Statistic: Maximum

Recommended threshold: 1.0

Threshold justification: When a status check fails, the value of this metric is 1. The threshold is set so that whenever the status check fails, the alarm is in ALARM state.

Period: 60

Datapoints to alarm: 10

Evaluation periods: 10

Comparison Operator: GREATER_THAN_OR_EQUAL_TO_THRESHOLD

Amazon EC2

CPUUtilization

Dimensions: InstanceId

Alarm description: This alarm helps to monitor the CPU utilization of an EC2 instance. Depending on the application, consistently high utilization levels might be normal. But if performance is degraded, and the application is not constrained by disk I/O, memory, or network resources, then a maxed-out CPU might indicate a resource bottleneck or application performance problems. High CPU utilization might indicate that an upgrade to a more CPU intensive instance is required. If detailed monitoring is enabled, you can change the period to 60 seconds instead of 300 seconds. For more information, see Enable or turn off detailed monitoring for your instances.

Intent: This alarm is used to detect high CPU utilization.

Statistic: Average

Recommended threshold: 80.0

Threshold justification: Typically, you can set the threshold for CPU utilization to 70-80%. However, you can adjust this value based on your acceptable performance level and workload characteristics. For some systems, consistently high CPU utilization may be normal and not indicate a problem, while for others, it may be cause of concern. Analyze historical CPU utilization data to identify the usage, find what CPU utilization is acceptable for your system, and set the threshold accordingly.

Period: 300

Datapoints to alarm: 3

Evaluation periods: 3

Comparison Operator: GREATER_THAN_THRESHOLD

StatusCheckFailed

Dimensions: InstanceId

Alarm description: This alarm helps to monitor both system status checks and instance status checks. If either type of status check fails, then this alarm should be in ALARM state.

Intent: This alarm is used to detect the underlying problems with instances, including both system status check failures and instance status check failures.

Statistic: Maximum

Recommended threshold: 1.0

Threshold justification: When a status check fails, the value of this metric is 1. The threshold is set so that whenever the status check fails, the alarm is in ALARM state.

Period: 300

Datapoints to alarm: 2

Evaluation periods: 2

Comparison Operator: GREATER_THAN_OR_EQUAL_TO_THRESHOLD

StatusCheckFailed_AttachedEBS

Dimensions: InstanceId

Alarm description: This alarm helps you monitor whether the Amazon EBS volumes attached to an instance are reachable and able to complete I/O operations. This status check detects underlying issues with the compute or Amazon EBS infrastructure such as the following:

- Hardware or software issues on the storage subsystems underlying the Amazon EBS volumes
- Hardware issues on the physical host that impact reachability of the Amazon EBS volumes
- Connectivity issues between the instance and Amazon EBS volumes

When the attached EBS status check fails, you can either wait for Amazon to resolve the issue, or you can take an action such as replacing the affected volumes or stopping and restarting the instance.

Intent: This alarm is used to detect unreachable Amazon EBS volumes attached to an instance. These can cause failures in I/O operations.

Statistic: Maximum

Recommended threshold: 1.0

Threshold justification: When a status check fails, the value of this metric is 1. The threshold is set so that whenever the status check fails, the alarm is in ALARM state.

Period: 60

Datapoints to alarm: 10

Evaluation periods: 10

Comparison Operator: GREATER_THAN_OR_EQUAL_TO_THRESHOLD

Amazon ElastiCache

CPUUtilization

Dimensions: CacheClusterId, CacheNodeId

Alarm description: This alarm helps to monitor the CPU utilization for the entire ElastiCache instance, including the database engine processes and other processes running on the instance. Amazon Elasticache supports two engine types: Memcached and Redis OSS. When you reach high CPU utilization on a Memcached node, you should consider scaling up your instance type or adding new cache nodes. For Redis OSS, if your main workload is from read requests, you should consider adding more read replicas to your cache cluster. If your main workload is from write requests, you should consider adding more shards to distribute the workload across more primary nodes if you're running in clustered mode, or scaling up your instance type if you're running Redis OSS in non-clustered mode.

Intent: This alarm is used to detect high CPU utilization of ElastiCache hosts. It is useful to get a broad view of the CPU usage across the entire instance, including non-engine processes.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: Set the threshold to the percentage that reflects a critical CPU utilization level for your application. For Memcached, the engine can use up to num_threads cores. For Redis OSS, the engine is largely single-threaded, but might use additional cores if available to accelerate I/O. In most cases, you can set the threshold to about 90% of your available CPU. Because Redis OSS is single-threaded, the actual threshold value should be calculated as a fraction of the node's total capacity.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

CurrConnections

Dimensions: CacheClusterId, CacheNodeId

Alarm description: This alarm detects high connection count, which might indicate heavy load or performance issues. A constant increase of `CurrConnections` might lead to exhaustion of the 65,000 available connections. It may indicate that connections improperly closed on the application side and were left established on the server side. You should consider using connection pooling or idle connection timeouts to limit the number of connections made to the cluster, or for Redis OSS, consider tuning tcp-keepalive on your cluster to detect and terminate potential dead peers.

Intent: The alarm helps you identify high connection counts that could impact the performance and stability of your ElastiCache cluster.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: The recommended threshold value for this alarm is highly dependent on the acceptable range of connections for your cluster. Review the capacity and the expected workload of your ElastiCache cluster and analyze the historical connection counts during regular usage to establish a baseline, and then select a threshold accordingly. Remember that each node can support up to 65,000 concurrent connections.

Period: 60

Datapoints to alarm: 10

Evaluation periods: 10

Comparison Operator: GREATER_THAN_THRESHOLD

DatabaseMemoryUsagePercentage

Dimensions: CacheClusterId

Alarm description: This alarm helps you monitor the memory utilization of your cluster. When your `DatabaseMemoryUsagePercentage` reaches 100%, the Redis OSS maxmemory policy is triggered and evictions might occur based on the policy selected. If no object in the cache matches the eviction policy, write operations fail. Some workloads expect or rely on evictions, but if not, you will need to increase the memory capacity of your cluster. You can scale your cluster out by adding more primary nodes, or scale it up by using a larger node type. Refer to Scaling ElastiCache for Redis OSS clusters for details.

Intent: This alarm is used to detect high memory utilization of your cluster so that you can avoid failures when writing to your cluster. It is useful to know when you'll need to scale up your cluster if your application does not expect to experience evictions.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: Depending on your application's memory requirements and the memory capacity of your ElastiCache cluster, you should set the threshold to the percentage that reflects the critical level of memory usage of the cluster. You can use historical memory usage data as reference for acceptable memory usage threshold.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

EngineCPUUtilization

Dimensions: CacheClusterId

Alarm description: This alarm helps to monitor the CPU utilization of a Redis OSS engine thread within the ElastiCache instance. Common reasons for high engine CPU are long-running commands that consume high CPU, a high number of requests, an increase of new client connection requests in a short time period, and high evictions when the cache doesn't have enough memory to hold new data. You should consider Scaling ElastiCache for Redis OSS clusters by adding more nodes or scaling up your instance type.

Intent: This alarm is used to detect high CPU utilization of the Redis OSS engine thread. It is useful if you want to monitor the CPU usage of the database engine itself.

Statistic: Average

Recommended threshold: 90.0

Threshold justification: Set the threshold to a percentage that reflects the critical engine CPU utilization level for your application. You can benchmark your cluster using your application and expected workload to correlate EngineCPUUtilization and performance as a reference, and then set the threshold accordingly. In most cases, you can set the threshold to about 90% of your available CPU.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

ReplicationLag

Dimensions: CacheClusterId

Alarm description: This alarm helps to monitor the replication health of your ElastiCache cluster. A high replication lag means that the primary node or the replica can't keep up the pace of the replication. If your write activity is too high, consider scaling your cluster out by adding more primary nodes, or scaling it up by using a larger node type. Refer to Scaling ElastiCache for Redis OSS clusters for details. If your read replicas are overloaded by the amount of read requests, consider adding more read replicas.

Intent: This alarm is used to detect a delay between data updates on the primary node and their synchronization to replica node. It helps to ensure data consistency of a read replica cluster node.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: Set the threshold according to your application's requirements and the potential impact of replication lag. You should consider your application's expected write rates and network conditions for the acceptable replication lag.

Period: 60

Datapoints to alarm: 15

Evaluation periods: 15

Comparison Operator: GREATER_THAN_THRESHOLD

Amazon ECS

The following are the recommended alarms for Amazon ECS.

CPUReservation

Dimensions: ClusterName

Alarm description: This alarm helps you detect a high CPU reservation of the ECS cluster. High CPU reservation might indicate that the cluster is running out of registered CPUs for the task. To troubleshoot, you can add more capacity, you can scale the cluster, or you can set up auto scaling.

Intent: The alarm is used to detect whether the total number of CPU units reserved by tasks on the cluster is reaching the total CPU units registered for the cluster. This helps you know when to scale up the cluster. Reaching the total CPU units for the cluster can result in running out of CPU for tasks. If you have EC2 capacity providers managed scaling turned on, or you have associated Fargate to capacity providers, then this alarm is not recommended.

Statistic: Average

Recommended threshold: 80.0

Threshold justification: Set the threshold for CPU reservation to 80%. Alternatively, you can choose a lower value based on cluster characteristics.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

CPUUtilization

Dimensions: ClusterName, ServiceName

Alarm description: This alarm helps you detect a high CPU utilization of the ECS service. If there is no ongoing ECS deployment, a maxed-out CPU utilization might indicate a resource bottleneck or application performance problems. To troubleshoot, you can increase the CPU limit.

Intent: This alarm is used to detect high CPU utilization for the Amazon ECS service. Consistent high CPU utilization can indicate a resource bottleneck or application performance problems.

Statistic: Average

Recommended threshold: 80.0

Threshold justification: The service metrics for CPU utilization might exceed 100% utilization. However, we recommend that you monitor the metric for high CPU utilization to avoid impacting other services. Set the threshold to about 80-85%. We recommend that you update your task definitions to reflect actual usage to prevent future issues with other services.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

EBSFilesystemUtilization

Dimensions: ClusterName, ServiceName

Alarm description: This alarm helps you detect high storage utilization of the Amazon EBS volume attached to Amazon ECS tasks. If the utilization of the Amazon EBS volume is consistently high, you can check the usage and increase the volume size for new tasks.

Intent: This alarm is used to detect high storage utilization of the Amazon EBS volumes attached to Amazon ECS tasks. Consistently high storage utilization can indicate that the Amazon EBS volume is full and it might lead to failure of the container.

Statistic: Average

Recommended threshold: 80.0

Threshold justification: You can set the threshold for Amazon EBS file system utilization to about 80%. You can adjust this value based on the acceptable storage utilization. For a read-only snapshot volume, a high utilization might indicate that the volume is right sized. For an active data volume, high storage utilization might indicate that the application is writing a large amount of data which may cause the container to fail if there is not enough capacity.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

MemoryReservation

Dimensions: ClusterName

Alarm description: This alarm helps you detect a high memory reservation of the Amazon ECS cluster. High memory reservation might indicate a resource bottleneck for the cluster. To troubleshoot, analyze the service task for performance to see if memory utilization of the task can be optimized. Also, you can register more memory or set up auto scaling.

Intent: The alarm is used to detect whether the total memory units reserved by tasks on the cluster is reaching the total memory units registered for the cluster. This can help you know when to scale up the cluster. Reaching the total memory units for the cluster can cause the cluster to be unable to launch new tasks. If you have EC2 capacity providers managed scaling turned on or you have associated Fargate to capacity providers, this alarm is not recommended.

Statistic: Average

Recommended threshold: 80.0

Threshold justification: Set the threshold for memory reservation to 80%. You can adjust this to a lower value based on cluster characteristics.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

MemoryUtilization

Dimensions: ClusterName, ServiceName

Alarm description: This alarm helps you detect a high memory utilization of the Amazon ECS service. If there is no ongoing Amazon ECS deployment, a maxed-out memory utilization might indicate a resource bottleneck or application performance problems. To troubleshoot, you can increase the memory limit.

Intent: This alarm is used to detect high memory utilization for the Amazon ECS service. Consistent high memory utilization can indicate a resource bottleneck or application performance problems.

Statistic: Average

Recommended threshold: 80.0

Threshold justification: The service metrics for memory utilization might exceed 100% utilization. However, we recommend that you monitor the metric for high memory utilization to avoid impacting other services. Set the threshold to about 80%. We recommend that you update your task definitions to reflect actual usage to prevent future issues with other services.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

HTTPCode_Target_5XX_Count

Dimensions: ClusterName, ServiceName

Alarm description: This alarm helps you detect a high server-side error count for the ECS service. This can indicate that there are errors that cause the server to be unable to serve requests. To troubleshoot, check your application logs.

Intent: This alarm is used to detect a high server-side error count for the ECS service.

Statistic: Sum

Recommended threshold: Depends on your situation

Threshold justification: Calculate the value of about 5% of the your average traffic and use this value as a starting point for the threshold. You can find the average traffic by using the `RequestCount` metric. You can also analyze historical data to determine the acceptable error rate for the application workload, and then tune the threshold accordingly. Frequently occurring 5XX errors need to be alarmed on. However, setting a very low value for the threshold can cause the alarm to be too sensitive.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

TargetResponseTime

Dimensions: ClusterName, ServiceName

Alarm description: This alarm helps you detect a high target response time for ECS service requests. This can indicate that there are problems that cause the service to be unable to serve requests in time. To troubleshoot, check the CPUUtilization metric to see if the service is running out of CPU, or check the CPU utilization of other downstream services that your service depends on.

Intent: This alarm is used to detect a high target response time for ECS service requests.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: The recommended threshold value for this alarm is highly dependent on your use case. Review the criticality and requirements of the target response time of the service and analyze the historical behavior of this metric to determine sensible threshold levels.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

Amazon ECS with Container Insights

The following are the recommended alarms for Amazon ECS with Container Insights.

EphemeralStorageUtilized

Dimensions: ClusterName, ServiceName

Alarm description: This alarm helps you detect high ephemeral storage utilized of the Fargate cluster. If ephemeral storage is consistently high, you can check ephemeral storage usage and increase the ephemeral storage.

Intent: This alarm is used to detect high ephemeral storage usage for the Fargate cluster. Consistent high ephemeral storage utilized can indicate that the disk is full and it might lead to failure of the container.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: Set the threshold to about 90% of the ephemeral storage size. You can adjust this value based on your acceptable ephemeral storage utilization of the Fargate cluster. For some systems, a consistently high ephemeral storage utilized might be normal, while for others, it might lead to failure of the container.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

RunningTaskCount

Dimensions: ClusterName, ServiceName

Alarm description: This alarm helps you detect a low running task count of the Amazon ECS service. If the running task count is too low, it can can indicate that the application can't handle the service load and it might lead to performance issues. If there is no running task, the Amazon ECS service might be unavailable or there might be deployment issues.

Intent: This alarm is used to detect whether the number of running tasks are too low. A consistent low running task count can indicate Amazon ECS service deployment or performance issues.

Statistic: Average

Recommended threshold: 0.0

Threshold justification: You can adjust the threshold based on the minimum running task count of the Amazon ECS service. If the running task count is 0, the Amazon ECS service will be unavailable.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: LESS_THAN_OR_EQUAL_TO_THRESHOLD

TaskCpuUtilization

Dimensions: ClusterName

Alarm description: This alarm helps you detect high CPU utilization of tasks in your Amazon ECS cluster. If task CPU utilization is consistently high, you might need to optimize your tasks or increase their CPU reservation.

Intent: This alarm is used to detect high CPU utilization for tasks in the Amazon ECS cluster. Consistent high CPU utilization can indicate that the tasks are under stress and might need more CPU resources or optimization to maintain performance.

Statistic: Average

Recommended threshold: 80.0

Threshold justification: Set the threshold to about 80% of the task's CPU reservation. You can adjust this value based on your acceptable CPU utilization for the tasks. For some workloads, consistently high CPU utilization might be normal, while for others, it might indicate performance issues or the need for more resources.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

TaskCpuUtilization

Dimensions: ClusterName, ServiceName

Alarm description: This alarm helps you detect high CPU utilization of tasks belonging to the Amazon ECS service. If task CPU utilization is consistently high, you might need to optimize your tasks or increase their CPU reservation.

Intent: This alarm is used to detect high CPU utilization for tasks belonging to the Amazon ECS service. Consistent high CPU utilization can indicate that the tasks are under stress and might need more CPU resources or optimization to maintain performance.

Statistic: Average

Recommended threshold: 80.0

Threshold justification: Set the threshold to about 80% of the task's CPU reservation. You can adjust this value based on your acceptable CPU utilization for the tasks. For some workloads, consistently high CPU utilization might be normal, while for others, it might indicate performance issues or the need for more resources.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

ContainerCpuUtilization

Dimensions: ClusterName

Alarm description: This alarm monitors the percentage of CPU units used by containers in your Amazon ECS cluster relative to their reserved CPU. It helps detect when containers are approaching their CPU limits based on the ContainerCpuUtilized/ContainerCpuReserved ratio.

Intent: This alarm detects when containers in the Amazon ECS cluster are using a high percentage of their reserved CPU capacity, calculated as `ContainerCpuUtilized/ContainerCpuReserved`. Sustained high values indicate containers are operating near their CPU limits and might need capacity adjustments.

Statistic: Average

Recommended threshold: 80.0

Threshold justification: Set the threshold to about 80% of the container's CPU utilization ratio. This provides an early warning when containers are approaching their CPU capacity limits while allowing for normal fluctuations in CPU usage. The threshold can be adjusted based on your workload characteristics and performance requirements.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

ContainerCpuUtilization

Dimensions: ClusterName, ServiceName

Alarm description: This alarm monitors the percentage of CPU units used by containers belonging to the Amazon ECS service relative to their reserved CPU. It helps detect when containers are approaching their CPU limits based on the ContainerCpuUtilized/ContainerCpuReserved ratio.

Intent: This alarm detects when containers belonging to the Amazon ECS service are using a high percentage of their reserved CPU capacity, calculated as ContainerCpuUtilized/ContainerCpuReserved. Sustained high values indicate containers are operating near their CPU limits and might need capacity adjustments.

Statistic: Average

Recommended threshold: 80.0

Threshold justification: Set the threshold to about 80% of the container's CPU utilization ratio. This provides an early warning when containers are approaching their CPU capacity limits while allowing for normal fluctuations in CPU usage. The threshold can be adjusted based on your workload characteristics and performance requirements.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

TaskEphemeralStorageUtilization

Dimensions: ClusterName

Alarm description: This alarm helps you detect high ephemeral storage utilization of tasks in your Amazon ECS cluster. If storage utilization is consistently high, you might need to optimize your storage usage or increase the storage reservation.

Intent: This alarm is used to detect high ephemeral storage utilization for tasks in the Amazon ECS cluster. Consistent high storage utilization can indicate that the task is running out of disk space and might need more storage resources or optimization to maintain proper operation.

Statistic: Average

Recommended threshold: 80.0

Threshold justification: Set the threshold to about 80% of the task's ephemeral storage reservation. You can adjust this value based on your acceptable storage utilization for the tasks. For some workloads, consistently high storage utilization might be normal, while for others, it might indicate potential disk space issues or the need for more storage.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

TaskEphemeralStorageUtilization

Dimensions: ClusterName, ServiceName

Alarm description: This alarm helps you detect high ephemeral storage utilization of tasks belonging to the Amazon ECS service. If storage utilization is consistently high, you might need to optimize your storage usage or increase the storage reservation.

Intent: This alarm is used to detect high ephemeral storage utilization for tasks belonging to the Amazon ECS service. Consistent high storage utilization can indicate that the task is running out of disk space and might need more storage resources or optimization to maintain proper operation.

Statistic: Average

Recommended threshold: 80.0

Threshold justification: Set the threshold to about 80% of the task's ephemeral storage reservation. You can adjust this value based on your acceptable storage utilization for the tasks. For some workloads, consistently high storage utilization might be normal, while for others, it might indicate potential disk space issues or the need for more storage.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

TaskMemoryUtilization

Dimensions: ClusterName

Alarm description: This alarm helps you detect high memory utilization of tasks in your Amazon ECS cluster. If memory utilization is consistently high, you might need to optimize your tasks or increase the memory reservation.

Intent: This alarm is used to detect high memory utilization for tasks in the Amazon ECS cluster. Consistent high memory utilization can indicate that the task is under memory pressure and might need more memory resources or optimization to maintain stability.

Statistic: Average

Recommended threshold: 80.0

Threshold justification: Set the threshold to about 80% of the task's memory reservation. You can adjust this value based on your acceptable memory utilization for the tasks. For some workloads, consistently high memory utilization might be normal, while for others, it might indicate memory pressure or the need for more resources.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

TaskMemoryUtilization

Dimensions: ClusterName, ServiceName

Alarm description: This alarm helps you detect high memory utilization of tasks belonging to the Amazon ECS service. If memory utilization is consistently high, you might need to optimize your tasks or increase the memory reservation.

Intent: This alarm is used to detect high memory utilization for tasks belonging to the Amazon ECS service. Consistent high memory utilization can indicate that the task is under memory pressure and might need more memory resources or optimization to maintain stability.

Statistic: Average

Recommended threshold: 80.0

Threshold justification: Set the threshold to about 80% of the task's memory reservation. You can adjust this value based on your acceptable memory utilization for the tasks. For some workloads, consistently high memory utilization might be normal, while for others, it might indicate memory pressure or the need for more resources.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

ContainerMemoryUtilization

Dimensions: ClusterName

Alarm description: This alarm helps you detect high memory utilization of containers in your Amazon ECS cluster. If memory utilization is consistently high, you might need to optimize your containers or increase the memory reservation.

Intent: This alarm is used to detect high memory utilization for containers in the Amazon ECS cluster. Consistent high memory utilization can indicate that the container is under memory pressure and might need more memory resources or optimization to maintain stability.

Statistic: Average

Recommended threshold: 80.0

Threshold justification: Set the threshold to about 80% of the container's memory reservation. You can adjust this value based on your acceptable memory utilization for the containers. For some workloads, consistently high memory utilization might be normal, while for others, it might indicate memory pressure or the need for more resources.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

ContainerMemoryUtilization

Dimensions: ClusterName, ServiceName

Alarm description: This alarm helps you detect high memory utilization of containers belonging to the Amazon ECS service. If memory utilization is consistently high, you might need to optimize your containers or increase the memory reservation.

Intent: This alarm is used to detect high memory utilization for containers belonging to the Amazon ECS service. Consistent high memory utilization can indicate that the container is under memory pressure and might need more memory resources or optimization to maintain stability.

Statistic: Average

Recommended threshold: 80.0

Threshold justification: Set the threshold to about 80% of the container's memory reservation. You can adjust this value based on your acceptable memory utilization for the containers. For some workloads, consistently high memory utilization might be normal, while for others, it might indicate memory pressure or the need for more resources.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

instance_filesystem_utilization

Dimensions: InstanceId, ContainerInstanceId, ClusterName

Alarm description: This alarm helps you detect a high file system utilization of the Amazon ECS cluster. If the file system utilization is consistently high, check the disk usage.

Intent: This alarm is used to detect high file system utilization for the Amazon ECS cluster. A consistent high file system utilization can indicate a resource bottleneck or application performance problems, and it might prevent running new tasks.

Statistic: Average

Recommended threshold: 90.0

Threshold justification: You can set the threshold for file system utilization to about 90-95%. You can adjust this value based on the acceptable file system capacity level of the Amazon ECS cluster. For some systems, a consistently high file system utilization might be normal and not indicate a problem, while for others, it might be a cause of concern and might lead to performance issues and prevent running new tasks.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

Amazon ECS with Container Insights with enhanced observability

The following are the recommended alarms for Amazon ECS with Container Insights with enhanced observability.

TaskCpuUtilization

Dimensions: ClusterName, ServiceName

Alarm description: This alarm helps you detect the total percentage of CPU units being used by a task.

Intent: This alarm is used to detect high task CPU utilization.

Statistic: Average

Recommended threshold: 80.0

Threshold justification: Typically, you can set the threshold for CPU utilization to 80%. However, you can adjust this value based on your acceptable performance level and workload characteristics. For some tasks, consistently high CPU utilization may be normal and not indicate a problem, while for others, it may be cause of concern. Analyze historical CPU utilization data to identify the usage, find what CPU utilization is acceptable for your tasks, and set the threshold accordingly.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

TaskMemoryUtilization

Dimensions: ClusterName, ServiceName

Alarm description: This alarm helps you detect the total percentage of memory being used by a task.

Intent: This alarm is used to detect high memory utilization.

Statistic: Average

Recommended threshold: 80.0

Threshold justification: Typically, you can set the threshold for memory utilization to 80%. However, you can adjust this value based on your acceptable performance level and workload characteristics. For some tasks, consistently high memory utilization may be normal and not indicate a problem, while for others, it may be cause of concern. Analyze historical memory utilization data to identify the usage, find what memory utilization is acceptable for your tasks, and set the threshold accordingly.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: LESS_THAN_OR_EQUAL_TO_THRESHOLD

ContainerCPUUtilization

Dimensions: ContainerName, ClusterName, ServiceName

Alarm description: This alarm helps you detect the total percentage of CPU units being used by a container.

Intent: This alarm is used to detect high task CPU utilization.

Statistic: Average

Recommended threshold: 80.0

Threshold justification: Typically, you can set the threshold for CPU utilization to 80%. However, you can adjust this value based on your acceptable performance level and workload characteristics. For some containers, consistently high CPU utilization may be normal and not indicate a problem, while for others, it may be cause of concern. Analyze historical CPU utilization data to identify the usage, find what CPU utilization is acceptable for your containers, and set the threshold accordingly.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

ContainerMemoryUtilization

Dimensions: ContainerName, ClusterName, ServiceName

Alarm description: This alarm helps you detect the total percentage of memory units being used by a container.

Intent: This alarm is used to detect high task memory utilization.

Statistic: Average

Recommended threshold: 80.0

Threshold justification: Typically, you can set the threshold for memory utilization to 80%. However, you can adjust this value based on your acceptable performance level and workload characteristics. For some containers, consistently high memory utilization may be normal and not indicate a problem, while for others, it may be cause of concern. Analyze historical memory utilization data to identify the usage, find what memory utilization is acceptable for your tasks, and set the threshold accordingly.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: LESS_THAN_OR_EQUAL_TO_THRESHOLD

TaskEBSfilesystemUtilization

Dimensions: ClusterName, ServiceName

Alarm description: This alarm helps you detect the total percentage of ephemeral storage being used by a task. .

Intent: This alarm is used to detect high Amazon EBS file system usage for a task.

Statistic: Average

Recommended threshold: 80.0

Threshold justification: Set the threshold to about 80% of the Amazon EBS file system size.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

TaskEphemeralStorageUtilization

Dimensions: ClusterName, ServiceName

Alarm description: This alarm helps you detect the total percentage of ephemeral storage being used by a task.

Intent: This alarm is used to detect high ephemeral storage usage for a task. Consistent high ephemeral storage utilized can indicate that the disk is full and it might lead to failure of the task.

Statistic: Average

Recommended threshold: 80.0

Threshold justification: Set the threshold to about 80% of the ephemeral storage size.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

Amazon EFS

PercentIOLimit

Dimensions: FileSystemId

Alarm description: This alarm helps in ensuring that the workload stays within the I/O limit available to the file system. If the metric reaches its I/O limit consistently, consider moving the application to a file system that uses Max I/O performance as mode. For troubleshooting, check clients that are connected to the file system and applications of the clients that throttles the file system.

Intent: This alarm is used to detect how close the file system is to reach the I/O limit of the General Purpose performance mode. Consistent high I/O percentage can be an indicator of the file system cannot scale with respect to I/O requests enough and the file system can be a resource bottleneck for the applications that use the file system.

Statistic: Average

Recommended threshold: 100.0

Threshold justification: When the file system reaches its I/O limit, it may respond to read and write requests slower. Therefore, it is recommended that the metric is monitored to avoid impacting applications that use the file system. The threshold can be set around 100%. However, this value can be adjusted to a lower value based on file system characteristics.

Period: 60

Datapoints to alarm: 15

Evaluation periods: 15

Comparison Operator: GREATER_THAN_OR_EQUAL_TO_THRESHOLD

BurstCreditBalance

Dimensions: FileSystemId

Alarm description: This alarm helps in ensuring that there is available burst credit balance for the file system usage. When there is no available burst credit, applications access to the the file system will be limited due to low throughput. If the metric drops to 0 consistently, consider changing the throughput mode to Elastic or Provisioned throughput mode.

Intent: This alarm is used to detect low burst credit balance of the file system. Consistent low burst credit balance can be an indicator of the slowing down in throughput and increase in I/O latency.

Statistic: Average

Recommended threshold: 0.0

Threshold justification: When the file system run out of burst credits and even if the baseline throughput rate is lower, EFS continues to provide a metered throughput of 1 MiBps to all file systems. However, it is recommended that the metric is monitored for low burst credit balance to avoid the file system acting as resource bottleneck for the applications. The threshold can be set around 0 bytes.

Period: 60

Datapoints to alarm: 15

Evaluation periods: 15

Comparison Operator: LESS_THAN_OR_EQUAL_TO_THRESHOLD

Amazon EKS with Container Insights

node_cpu_utilization

Dimensions: ClusterName

Alarm description: This alarm helps to detect high CPU utilization in worker nodes of the EKS cluster. If the utilization is consistently high, it might indicate a need for replacing your worker nodes with instances that have greater CPU or a need to scale the system horizontally.

Intent: This alarm helps to monitor the CPU utilization of the worker nodes in the EKS cluster so that the system performance doesn't degrade.

Statistic: Maximum

Recommended threshold: 80.0

Threshold justification: It is recommended to set the threshold at less than or equal to 80% to allow enough time to debug the issue before the system starts seeing impact.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

node_filesystem_utilization

Dimensions: ClusterName

Alarm description: This alarm helps to detect high file system utilization in the worker nodes of the EKS cluster. If the utilization is consistently high, you might need to update your worker nodes to have larger disk volume, or you might need to scale horizontally.

Intent: This alarm helps to monitor the filesystem utilization of the worker nodes in the EKS cluster. If the utilization reaches 100%, it can lead to application failure, disk I/O bottlenecks, pod eviction, or the node to become unresponsive entirely.

Statistic: Maximum

Recommended threshold: Depends on your situation

Threshold justification: If there's sufficient disk pressure (meaning that the disk is getting full), nodes are marked as not healthy, and the pods are evicted from the node. Pods on a node with disk pressure are evicted when the available file system is lower than the eviction thresholds set on the kubelet. Set the alarm threshold so that you have enough time to react before the node is evicted from the cluster.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

node_memory_utilization

Dimensions: ClusterName

Alarm description: This alarm helps in detecting high memory utilization in worker nodes of the EKS cluster. If the utilization is consistently high, it might indicate a need to scale the number of pod replicas, or optimize your application.

Intent: This alarm helps to monitor the memory utilization of the worker nodes in the EKS cluster so that the system performance doesn't degrade.

Statistic: Maximum

Recommended threshold: 80.0

Threshold justification: It is recommended to set the threshold at less than or equal to 80% to allow having enough time to debug the issue before the system starts seeing impact.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

pod_cpu_utilization_over_pod_limit

Dimensions: ClusterName, Namespace, Service

Alarm description: This alarm helps in detecting high CPU utilization in pods of the EKS cluster. If the utilization is consistently high, it might indicate a need to increase the CPU limit for the affected pod.

Intent: This alarm helps to monitor the CPU utilization of the pods belonging to a Kubernetes Service in the EKS cluster, so that you can quickly identify if a service's pod is consuming higher CPU than expected.

Statistic: Maximum

Recommended threshold: 80.0

Threshold justification: It is recommended to set the threshold at less than or equal to 80% to allow having enough time to debug the issue before the system starts seeing impact.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

pod_memory_utilization_over_pod_limit

Dimensions: ClusterName, Namespace, Service

Alarm description: This alarm helps in detecting high memory utilization in pods of the EKS cluster. If the utilization is consistently high, it might indicate a need to increase the memory limit for the affected pod.

Intent: This alarm helps to monitor the memory utilization of the pods in the EKS cluster so that the system performance doesn't degrade.

Statistic: Maximum

Recommended threshold: 80.0

Threshold justification: It is recommended to set the threshold at less than or equal to 80% to allow having enough time to debug the issue before the system starts seeing impact.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

Amazon EventBridge Scheduler

TargetErrorThrottledCount

Dimensions: None

Alarm description: This alarm helps you identify target throttling. To avoid target throttling error, consider configuring flexible time windows to spread your invocation load or increasing limits with the target service.

Intent: This alarm is used to detect target throttling errors, which can cause schedule delays.

Statistic: Sum

Recommended threshold: 0.0

Threshold justification: If the target throttling error is consistently greater than 0, schedule delivery is delayed. For some systems, target throttling errors for a brief period of time might be normal, while for others, it might be a cause of concern. Set this alarm's threshold, `datapointsToAlarm`, and `evaluationPeriods` accordingly.

Period: 60

Datapoints to alarm: 15

Evaluation periods: 15

Comparison Operator: GREATER_THAN_THRESHOLD

InvocationThrottleCount

Dimensions: None

Alarm description: This alarm helps you identify invocation throttling by Amazon EventBridge Scheduler. To avoid invocation throttling errors, consider configuring flexible time windows to spread your invocation load or increasing invocations throttle limit.

Intent: This alarm is used to detect Amazon EventBridge Scheduler invocation throttling errors, which can cause schedule delays.

Statistic: Sum

Recommended threshold: 0.0

Threshold justification: If the invocation throttling is consistently greater than 0, schedule delivery is delayed. For some systems, invocation throttling errors for a brief period of time might be normal, while for others, it might be a cause of concern. Set this alarm's threshold, `datapointsToAlarm`, and `evaluationPeriods` accordingly.

Period: 60

Datapoints to alarm: 15

Evaluation periods: 15

Comparison Operator: GREATER_THAN_THRESHOLD

InvocationDroppedCount

Dimensions: None

Alarm description: This alarm helps you identify invocations dropped by Amazon EventBridge Scheduler. Consider investigating by configuring a DLQ for the schedule.

Intent: This alarm is used to detect dropped invocations by Amazon EventBridge Scheduler. If you have configured a DLQ correctly on all of your schedules, dropped invocations will appear in the DLQ and you can skip setting up this alarm.

Statistic: Sum

Recommended threshold: 0.0

Threshold justification: Set the threshold to 0 to detect dropped invocations.

Period: 60

Datapoints to alarm: 1

Evaluation periods: 1

Comparison Operator: GREATER_THAN_THRESHOLD

InvocationsFailedToBeSentToDeadLetterCount

Dimensions: None

Alarm description: This alarm helps you identify invocations that were failed to be sent to the configured DLQ by Amazon EventBridge Scheduler. If the metric is consistently greater than 0, modify your DLQ configuration to resolve the issue. Use `InvocationsFailedToBeSentToDeadLetterCount`_metrics to determine the issue.

Intent: This alarm is used to detect invocations failed to be sent to the configured DLQ by Amazon EventBridge Scheduler.

Statistic: Sum

Recommended threshold: 0.0

Threshold justification: Set the threshold to 0 to detect any invocations that were failed to be sent to the configured DLQ. Retryable errors also show up in this metric, so `datapointsToAlarm` for this alarm has been set to 15.

Period: 60

Datapoints to alarm: 15

Evaluation periods: 15

Comparison Operator: GREATER_THAN_THRESHOLD

Amazon Kinesis Data Streams

GetRecords.IteratorAgeMilliseconds

Dimensions: StreamName

Alarm description: This alarm can detect if iterator maximum age is too high. For real-time data processing applications, configure data retention according to tolerance of the delay. This is usually within minutes. For applications that process historic data, use this metric to monitor catchup speed. A quick solution to stop data loss is to increase the retention period while you troubleshoot the issue. You can also increase the number of workers processing records in your consumer application. The most common causes for gradual iterator age increase are insufficient physical resources or record processing logic that has not scaled with an increase in stream throughput. See link for more details.

Intent: This alarm is used to detect if data in your stream is going to expire because of being preserved too long or because record processing is too slow. It helps you avoid data loss after reaching 100% of the stream retention time.

Statistic: Maximum

Recommended threshold: Depends on your situation

Threshold justification: The recommended threshold value for this alarm is highly dependent on the stream retention period and tolerance of processing delay for the records. Review your requirements and analyze historical trends, and then set the threshold to the number of milliseconds that represents a critical processing delay. If an iterator's age passes 50% of the retention period (by default, 24 hours, configurable up to 365 days), there is a risk for data loss because of record expiration. You can monitor the metric to make sure that none of your shards ever approach this limit.

Period: 60

Datapoints to alarm: 15

Evaluation periods: 15

Comparison Operator: GREATER_THAN_THRESHOLD

GetRecords.Success

Dimensions: StreamName

Alarm description: This metric increments whenever your consumers successfully read data from your stream. `GetRecords` doesn't return any data when it throws an exception. The most common exception is `ProvisionedThroughputExceededException` because request rate for the stream is too high, or because available throughput is already served for the given second. Reduce the frequency or size of your requests. For more information, see Streams [Limits](#) in the Amazon Kinesis Data Streams Developer Guide, and [Error Retries and Exponential Backoff in Amazon](#).

Intent: This alarm can detect if the retrieval of records from the stream by consumers is failing. By setting an alarm on this metric, you can proactively detect any issues with data consumption, such as increased error rates or a decline in successful retrievals. This allows you to take timely actions to resolve potential problems and maintain a smooth data processing pipeline.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: Depending on the importance of retrieving records from the stream, set the threshold based on your application’s tolerance for failed records. The threshold should be the corresponding percentage of successful operations. You can use historical GetRecords metric data as reference for the acceptable failure rate. You should also consider retries when setting the threshold because failed records can be retried. This helps to prevent transient spikes from triggering unnecessary alerts.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: LESS_THAN_THRESHOLD

PutRecord.Success

Dimensions: StreamName

Alarm description: This alarm detects when the number of failed `PutRecord` operations breaches the threshold. Investigate the data producer logs to find the root causes of the failures. The most common reason is insufficient provisioned throughput on the shard that caused the `ProvisionedThroughputExceededException`. It happens because the request rate for the stream is too high, or the throughput attempted to be ingested into the shard is too high. Reduce the frequency or size of your requests. For more information, see Streams Limits and Error Retries and Exponential Backoff in Amazon.

Intent: This alarm can detect if ingestion of records into the stream is failing. It helps you identify issues in writing data to the stream. By setting an alarm on this metric, you can proactively detect any issues of producers in publishing data to the stream, such as increased error rates or a decrease in successful records being published. This enables you to take timely actions to address potential problems and maintain a reliable data ingestion process.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: Depending on the importance of data ingestion and processing to your service, set the threshold based on your application’s tolerance for failed records. The threshold should be the corresponding percentage of successful operations. You can use historical PutRecord metric data as reference for the acceptable failure rate. You should also consider retries when setting the threshold because failed records can be retried.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: LESS_THAN_THRESHOLD

PutRecords.FailedRecords

Dimensions: StreamName

Alarm description: This alarm detects when the number of failed `PutRecords` exceeds the threshold. Kinesis Data Streams attempts to process all records in each `PutRecords` request, but a single record failure does not stop the processing of subsequent records. The main reason for these failures is exceeding the throughput of a stream or an individual shard. Common causes are traffic spikes and network latencies that cause records to arrive to the stream unevenly. You should detect unsuccessfully processed records and retry them in a subsequent call. Refer to [Handling Failures When Using PutRecords] for more details.

Intent: This alarm can detect consistent failures when using batch operation to put records to your stream. By setting an alarm on this metric, you can proactively detect an increase in failed records, enabling you to take timely actions to address the underlying problems and ensure a smooth and reliable data ingestion process.

Statistic: Sum

Recommended threshold: Depends on your situation

Threshold justification: Set the threshold to the number of failed records reflecting the tolerance of the the application for failed records. You can use historical data as reference for the acceptable failure value. You should also consider retries when setting the threshold because failed records can be retried in subsequent PutRecords calls.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

ReadProvisionedThroughputExceeded

Dimensions: StreamName

Alarm description: The alarm tracks the number of records that result in read throughput capacity throttling. If you find that you are being consistently throttled, you should consider adding more shards to your stream to increase your provisioned read throughput. If there is more than one consumer application running on the stream, and they share the `GetRecords` limit, we recommend that you register new consumer applications via Enhanced Fan-Out. If adding more shards does not lower the number of throttles, you may have a “hot” shard that is being read from more than other shards are. Enable enhanced monitoring, find the “hot” shard, and split it.

Intent: This alarm can detect if consumers are throttled when they exceed your provisioned read throughput (determined by the number of shards you have). In that case, you won’t be able to read from the stream, and the stream can start backing up.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: Usually throttled requests can be retried and hence setting the threshold to zero makes the alarm too sensitive. However, consistent throttling can impact reading from the stream and should trigger the alarm. Set the threshold to a percentage according to the throttled requests for the application and retry configurations.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

SubscribeToShardEvent.MillisBehindLatest

Dimensions: StreamName, ConsumerName

Alarm description: This alarm detects when the delay of record processing in the application breaches the threshold. Transient problems such as API operation failures to a downstream application can cause a sudden increase in the metric. You should investigate if they consistently happen. A common cause is the consumer is not processing records fast enough because of insufficient physical resources or record processing logic that has not scaled with an increase in stream throughput. Blocking calls in critical path is often the cause of slowdowns in record processing. You can increase your parallelism by increasing the number of shards. You should also confirm underlying processing nodes have sufficient physical resources during peak demand.

Intent: This alarm can detect delay in the subscription to shard event of the stream. This indicates a processing lag and can help identify potential issues with the consumer application's performance or the overall stream's health. When the processing lag becomes significant, you should investigate and address any bottlenecks or consumer application inefficiencies to ensure real-time data processing and minimize data backlog.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: The recommended threshold value for this alarm is highly dependent on the delay that your application can tolerate. Review your application's requirements and analyze historical trends, and then select a threshold accordingly. When the SubscribeToShard call succeeds, your consumer starts receiving SubscribeToShardEvent events over the persistent connection for up to 5 minutes, after which time you need to call SubscribeToShard again to renew the subscription if you want to continue to receive records.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

WriteProvisionedThroughputExceeded

Dimensions: StreamName

Alarm description: This alarm detects when the number of records resulting in write throughput capacity throttling reached the threshold. When your producers exceed your provisioned write throughput (determined by the number of shards you have), they are throttled and you won’t be able to put records to the stream. To address consistent throttling, you should consider adding shards to your stream. This raises your provisioned write throughput and prevents future throttling. You should also consider partition key choice when ingesting records. Random partition key is preferred because it spreads records evenly across the shards of the stream, whenever possible.

Intent: This alarm can detect if your producers are being rejected for writing records because of throttling of the stream or shard. If your stream is in Provisioned mode, then setting this alarm helps you proactively take actions when the data stream reaches its limits, allowing you to optimize the provisioned capacity or take appropriate scaling actions to avoid data loss and maintain smooth data processing.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: Usually throttled requests can be retried, so setting the threshold to zero makes the alarm too sensitive. However, consistent throttling can impact writing to the stream, and you should set the alarm threshold to detect this. Set the threshold to a percentage according to the throttled requests for the application and retry configurations.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

Lambda

ClaimedAccountConcurrency

Dimensions: None

Alarm description: This alarm helps to monitor if the concurrency of your Lambda functions is approaching the Region-level concurrency limit of your account. A function starts to be throttled if it reaches the concurrency limit. You can take the following actions to avoid throttling.

1. Request a concurrency increase in this Region.
2. Identify and reduce any unused reserved concurrency or provisioned concurrency.

3. Identify performance issues in the functions to improve the speed of processing and therefore improve throughput.
4. Increase the batch size of the functions, so that more messages are processed by each function invocation.

Intent: This alarm can proactively detect if the concurrency of your Lambda functions is approaching the Region-level concurrency quota of your account, so that you can act on it. Functions are throttled if `ClaimedAccountConcurrency` reaches the Region-level concurrency quota of the account. If you are using Reserved Concurrency (RC) or Provisioned Concurrency (PC), this alarm gives you more visibility on concurrency utilization than an alarm on `ConcurrentExecutions` would.

Statistic: Maximum

Recommended threshold: Depends on your situation

Threshold justification: You should calculate the value of about 90% of the concurrency quota set for the account in the Region, and use the result as the threshold value. By default, your account has a concurrency quota of 1,000 across all functions in a Region. However, you should check the quota of your account from the Service Quotas dashboard.

Period: 60

Datapoints to alarm: 10

Evaluation periods: 10

Comparison Operator: GREATER_THAN_THRESHOLD

Errors

Dimensions: FunctionName

Alarm description: This alarm detects high error counts. Errors includes the exceptions thrown by the code as well as exceptions thrown by the Lambda runtime. You can check the logs related to the function to diagnose the issue.

Intent: The alarm helps detect high error counts in function invocations.

Statistic: Sum

Recommended threshold: Depends on your situation

Threshold justification: Set the threshold to a number greater than zero. The exact value can depend on the tolerance for errors in your application. Understand the criticality of the invocations that the function is handling. For some applications, any error might be unacceptable, while other applications might allow for a certain margin of error.

Period: 60

Datapoints to alarm: 3

Evaluation periods: 3

Comparison Operator: GREATER_THAN_THRESHOLD

Throttles

Dimensions: FunctionName

Alarm description: This alarm detects a high number of throttled invocation requests. Throttling occurs when there is no concurrency is available for scale up. There are several approaches to resolve this issue. 1) Request a concurrency increase from Amazon Support in this Region. 2) Identify performance issues in the function to improve the speed of processing and therefore improve throughput. 3) Increase the batch size of the function, so that more messages are processed by each function invocation.

Intent: The alarm helps detect a high number of throttled invocation requests for a Lambda function. It is important to know if requests are constantly getting rejected due to throttling and if you need to improve Lambda function performance or increase concurrency capacity to avoid constant throttling.

Statistic: Sum

Recommended threshold: Depends on your situation

Threshold justification: Set the threshold to a number greater than zero. The exact value of the threshold can depend on the tolerance of the application. Set the threshold according to its usage and scaling requirements of the function.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_OR_EQUAL_TO_THRESHOLD

Duration

Dimensions: FunctionName

Alarm description: This alarm detects long duration times for processing an event by a Lambda function. Long durations might be because of changes in function code making the function take longer to execute, or the function's dependencies taking longer.

Intent: This alarm can detect a long running duration of a Lambda function. High runtime duration indicates that a function is taking a longer time for invocation, and can also impact the concurrency capacity of invocation if Lambda is handling a higher number of events. It is critical to know if the Lambda function is constantly taking longer execution time than expected.

Statistic: p90

Recommended threshold: Depends on your situation

Threshold justification: The threshold for the duration depends on your application and workloads and your performance requirements. For high-performance requirements, set the threshold to a shorter time to see if the function is meeting expectations. You can also analyze historical data for duration metrics to see the if the time taken matches the performance expectation of the function, and then set the threshold to a longer time than the historical average. Make sure to set the threshold lower than the configured function timeout.

Period: 60

Datapoints to alarm: 15

Evaluation periods: 15

Comparison Operator: GREATER_THAN_THRESHOLD

ConcurrentExecutions

Dimensions: FunctionName

Alarm description: This alarm helps to monitor if the concurrency of the function is approaching the Region-level concurrency limit of your account. A function starts to be throttled if it reaches the concurrency limit. You can take the following actions to avoid throttling.

1. Request a concurrency increase in this Region.

2. Identify performance issues in the functions to improve the speed of processing and therefore improve throughput.
3. Increase the batch size of the functions, so that more messages are processed by each function invocation.

To get better visibility on reserved concurrency and provisioned concurrency utilization, set an alarm on the new metric `ClaimedAccountConcurrency` instead.

Intent: This alarm can proactively detect if the concurrency of the function is approaching the Region-level concurrency quota of your account, so that you can act on it. A function is throttled if it reaches the Region-level concurrency quota of the account.

Statistic: Maximum

Recommended threshold: Depends on your situation

Threshold justification: Set the threshold to about 90% of the concurrency quota set for the account in the Region. By default, your account has a concurrency quota of 1,000 across all functions in a Region. However, you can check the quota of your account, as it can be increased by contacting Amazon support.

Period: 60

Datapoints to alarm: 10

Evaluation periods: 10

Comparison Operator: GREATER_THAN_THRESHOLD

Lambda Insights

We recommend setting best-practice alarms for the following Lambda Insights metrics.

memory_utilization

Dimensions: function_name

Alarm description: This alarm is used to detect if the memory utilization of a lambda function is approaching the configured limit. For troubleshooting, you can try to 1) Optimize your code. 2) Rightly size your memory allocation by accurately estimating the memory requirements. You can refer to Lambda Power Tuning for the same. 3) Use connection pooling. Refer to Using

Amazon RDS Proxy with Lambda for the connection pooling for RDS database. 4) You can also consider designing your functions to avoid storing large amounts of data in memory between invocations.

Intent: This alarm is used to detect if the memory utilization for the Lambda function is approaching the configured limit.

Statistic: Average

Threshold Suggestion: 90.0

Threshold Justification: Set the threshold to 90% to get an alert when memory utilization exceeds 90% of the allocated memory. You can adjust this to a lower value if you have a concern for the workload for memory utilization. You can also check the historical data for this metric and set the threshold accordingly.

Period: 60

Datapoints to alarm: 10

Evaluation Periods: 10

ComparisonOperator: GREATER_THAN_THRESHOLD

Amazon VPC (`AWS/NATGateway`)

ErrorPortAllocation

Dimensions: NatGatewayId

Alarm description: This alarm helps to detect when the NAT Gateway is unable to allocate ports to new connections. To resolve this issue, see Resolve port allocation errors on NAT Gateway.

Intent: This alarm is used to detect if the NAT gateway could not allocate a source port.

Statistic: Sum

Recommended threshold: 0.0

Threshold justification: If the value of ErrorPortAllocation is greater than zero, that means too many concurrent connections to a single popular destination are open through NATGateway.

Period: 60

Datapoints to alarm: 15

Evaluation periods: 15

Comparison Operator: GREATER_THAN_THRESHOLD

PacketsDropCount

Dimensions: NatGatewayId

Alarm description: This alarm helps to detect when packets are dropped by NAT Gateway. This might happen because of an issue with NAT Gateway, so check [Amazon service health dashboard](#) for the status of Amazon NAT Gateway in your Region. This can help you correlate the network issue related to traffic using NAT gateway.

Intent: This alarm is used to detect if packets are being dropped by NAT Gateway.

Statistic: Sum

Recommended threshold: Depends on your situation

Threshold justification: You should calculate the value of 0.01 percent of the total traffic on the NAT Gateway and use that result as the threshold value. Use historical data of the traffic on NAT Gateway to determine the threshold.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

Amazon Private Link (`AWS/PrivateLinkEndpoints`)

PacketsDropped

Dimensions: VPC Id, VPC Endpoint Id, Endpoint Type, Subnet Id, Service Name

Alarm description: This alarm helps to detect if the endpoint or endpoint service is unhealthy by monitoring the number of packets dropped by the endpoint. Note that packets larger than

8500 bytes that arrive at the VPC endpoint are dropped. For troubleshooting, see connectivity problems between an interface VPC endpoint and an endpoint service.

Intent: This alarm is used to detect if the endpoint or endpoint service is unhealthy.

Statistic: Sum

Recommended threshold: Depends on your situation

Threshold justification: Set the threshold according to the use case. If you want to be aware of the unhealthy status of the endpoint or endpoint service, you should set the threshold low so that you get a chance to fix the issue before a huge data loss. You can use historical data to understand the tolerance for dropped packets and set the threshold accordingly.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

Amazon Private Link (`AWS/PrivateLinkServices`)

RstPacketsSent

Dimensions: Service Id, Load Balancer Arn, Az

Alarm description: This alarm helps you detect unhealthy targets of an endpoint service based on the number of reset packets that are sent to endpoints. When you debug connection errors with a consumer of your service, you can validate whether the service is resetting connections with the RstPacketsSent metric, or if something else is failing on the network path.

Intent: This alarm is used to detect unhealthy targets of an endpoint service.

Statistic: Sum

Recommended threshold: Depends on your situation

Threshold justification: The threshold depends on the use case. If your use case can tolerate targets being unhealthy, you can set the threshold high. If the use case can’t tolerate unhealthy targets you can set the threshold very low.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

Amazon RDS

CPUUtilization

Dimensions: DBInstanceIdentifier

Alarm description: This alarm helps to monitor consistent high CPU utilization. CPU utilization measures non-idle time. Consider using Enhanced Monitoring or Performance Insights to review which wait time is consuming the most of the CPU time (`guest`, `irq`, `wait`, `nice`, and so on) for MariaDB, MySQL, Oracle, and PostgreSQL. Then evaluate which queries consume the highest amount of CPU. If you can't tune your workload, consider moving to a larger DB instance class.

Intent: This alarm is used to detect consistent high CPU utilization in order to prevent very high response time and time-outs. If you want to check micro-bursting of CPU utilization you can set a lower alarm evaluation time.

Statistic: Average

Recommended threshold: 90.0

Threshold justification: Random spikes in CPU consumption might not hamper database performance, but sustained high CPU can hinder upcoming database requests. Depending on the overall database workload, high CPU at your RDS/Aurora instance can degrade the overall performance.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

DatabaseConnections

Dimensions: DBInstanceIdentifier

Alarm description: This alarm detects a high number of connections. Review existing connections and terminate any that are in `sleep` state or that are improperly closed. Consider using connection pooling to limit the number of new connections. Alternatively, increase the DB instance size to use a class with more memory and hence a higher default value for `max_connections` or increase the `max_connections` value in RDS and Aurora MySQL and PostgreSQL for the current class if it can support your workload.

Intent: This alarm is used to help prevent rejected connections when the maximum number of DB connections is reached. This alarm is not recommended if you frequently change DB instance class, because doing so changes the memory and default maximum number of connections.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: The number of connections allowed depends on the size of your DB instance class and database engine-specific parameters related to processes/connections. You should calculate a value between 90-95% of the maximum number of connections for your database and use that result as the threshold value.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

EBSByteBalance%

Dimensions: DBInstanceIdentifier

Alarm description: This alarm helps to monitor a low percentage of throughput credits remaining. For troubleshooting, check latency problems in RDS.

Intent: This alarm is used to detect a low percentage of throughput credits remaining in the burst bucket. Low byte balance percentage can cause throughput bottleneck issues. This alarm is not recommended for Aurora PostgreSQL instances.

Statistic: Average

Recommended threshold: 10.0

Threshold justification: A throughput credit balance below 10% is considered to be poor and you should set the threshold accordingly. You can also set a lower threshold if your application can tolerate a lower throughput for the workload.

Period: 60

Datapoints to alarm: 3

Evaluation periods: 3

Comparison Operator: LESS_THAN_THRESHOLD

EBSIOBalance%

Dimensions: DBInstanceIdentifier

Alarm description: This alarm helps to monitor low percentage of IOPS credits remaining. For troubleshooting, see latency problems in RDS.

Intent: This alarm is used to detect a low percentage of I/O credits remaining in the burst bucket. Low IOPS balance percentage can cause IOPS bottleneck issues. This alarm is not recommended for Aurora instances.

Statistic: Average

Recommended threshold: 10.0

Threshold justification: An IOPS credits balance below 10% is considered to be poor and you can set the threshold accordingly. You can also set a lower threshold, if your application can tolerate a lower IOPS for the workload.

Period: 60

Datapoints to alarm: 3

Evaluation periods: 3

Comparison Operator: LESS_THAN_THRESHOLD

FreeableMemory

Dimensions: DBInstanceIdentifier

Alarm description: This alarm helps to monitor low freeable memory which can mean that there is a spike in database connections or that your instance may be under high memory pressure. Check for memory pressure by monitoring the CloudWatch metrics for `SwapUsage`\`in addition to `FreeableMemory`. If the instance memory consumption is frequently too high, this indicates that you should check your workload or upgrade your instance class. For Aurora reader DB instance, consider adding additional reader DB instances to the cluster. For information about troubleshooting Aurora, see freeable memory issues.

Intent: This alarm is used to help prevent running out of memory which can result in rejected connections.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: Depending on the workload and instance class, different values for the threshold can be appropriate. Ideally, available memory should not go below 25% of total memory for prolonged periods. For Aurora, you can set the threshold close to 5%, because the metric approaching 0 means that the DB instance has scaled up as much as it can. You can analyze the historical behavior of this metric to determine sensible threshold levels.

Period: 60

Datapoints to alarm: 15

Evaluation periods: 15

Comparison Operator: LESS_THAN_THRESHOLD

FreeLocalStorage

Dimensions: DBInstanceIdentifier

Alarm description: This alarm helps to monitor low free local storage. Aurora PostgreSQL-Compatible Edition uses local storage for storing error logs and temporary files. Aurora MySQL uses local storage for storing error logs, general logs, slow query logs, audit logs, and non-InnoDB temporary tables. These local storage volumes are backed by Amazon EBS Store and can be extended by using a larger DB instance class. For troubleshooting, check Aurora PostgreSQL-Compatible and MySQL-Compatible.

Intent: This alarm is used to detect how close the Aurora DB instance is to reaching the local storage limit, if you do not use Aurora Serverless v2 or higher. Local storage can reach capacity when you store non-persistent data, such as temporary table and log files, in the local storage. This alarm can prevent an out-of-space error that occurs when your DB instance runs out of local storage.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: You should calculate about 10%-20% of the amount of storage available based on velocity and trend of volume usage, and then use that result as the threshold value to proactively take action before the volume reaches its limit.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: LESS_THAN_THRESHOLD

FreeStorageSpace

Dimensions: DBInstanceIdentifier

Alarm description: This alarm watches for a low amount of available storage space. Consider scaling up your database storage if you frequently approach storage capacity limits. Include some buffer to accommodate unforeseen increases in demand from your applications. Alternatively, consider enabling RDS storage auto scaling. Additionally, consider freeing up more space by deleting unused or outdated data and logs. For further information, check RDS run out of storage document and PostgreSQL storage issues document.

Intent: This alarm helps prevent storage full issues. This can prevent downtime that occurs when your database instance runs out of storage. We do not recommend using this alarm if you have storage auto scaling enabled, or if you frequently change the storage capacity of the database instance.

Statistic: Minimum

Recommended threshold: Depends on your situation

Threshold justification: The threshold value will depend on the currently allocated storage space. Typically, you should calculate the value of 10 percent of the allocated storage space and use that result as the threshold value.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: LESS_THAN_THRESHOLD

MaximumUsedTransactionIDs

Dimensions: DBInstanceIdentifier

Alarm description: This alarm helps prevent transaction ID wraparound for PostgreSQL. Refer to the troubleshooting steps in this blog to investigate and resolve the issue. You can also refer to this blog to familiarize yourself further with autovacuum concepts, common issues and best practices.

Intent: This alarm is used to help prevent transaction ID wraparound for PostgreSQL.

Statistic: Average

Recommended threshold: 1.0E9

Threshold justification: Setting this threshold to 1 billion should give you time to investigate the problem. The default autovacuum_freeze_max_age value is 200 million. If the age of the oldest transaction is 1 billion, autovacuum is having a problem keeping this threshold below the target of 200 million transaction IDs.

Period: 60

Datapoints to alarm: 1

Evaluation periods: 1

Comparison Operator: GREATER_THAN_THRESHOLD

ReadLatency

Dimensions: DBInstanceIdentifier

Alarm description: This alarm helps to monitor high read latency. If storage latency is high, it's because the workload is exceeding resource limits. You can review I/O utilization relative to instance and allocated storage configuration. Refer to troubleshoot the latency of Amazon EBS volumes caused by an IOPS bottleneck. For Aurora, you can switch to an instance class that has I/O-Optimized storage configuration. See Planning I/O in Aurora for guidance.

Intent: This alarm is used to detect high read latency. Database disks normally have a low read/write latency, but they can have issues that can cause high latency operations.

Statistic: p90

Recommended threshold: Depends on your situation

Threshold justification: The recommended threshold value for this alarm is highly dependent on your use case. Read latencies higher than 20 milliseconds are likely a cause for investigation. You can also set a higher threshold if your application can have higher latency for read operations. Review the criticality and requirements of read latency and analyze the historical behavior of this metric to determine sensible threshold levels.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

ReplicaLag

Dimensions: DBInstanceIdentifier

Alarm description: This alarm helps you understand the number of seconds a replica is behind the primary instance. A PostgreSQL Read Replica reports a replication lag of up to five minutes if there are no user transactions occurring on the source database instance. When the ReplicaLag metric reaches 0, the replica has caught up to the primary DB instance. If the ReplicaLag metric returns -1, then replication is currently not active. For guidance related to RDS PostgreSQL, see replication best practices and for troubleshooting `ReplicaLag` and related errors, see troubleshooting ReplicaLag.

Intent: This alarm can detect the replica lag which reflects the data loss that could happen in case of a failure of the primary instance. If the replica gets too far behind the primary and the primary fails, the replica will be missing data that was in the primary instance.

Statistic: Maximum

Recommended threshold: 60.0

Threshold justification: Typically, the acceptable lag depends on the application. We recommend no more than 60 seconds.

Period: 60

Datapoints to alarm: 10

Evaluation periods: 10

Comparison Operator: GREATER_THAN_THRESHOLD

WriteLatency

Dimensions: DBInstanceIdentifier

Alarm description: This alarm helps to monitor high write latency. If storage latency is high, it's because the workload is exceeding resource limits. You can review I/O utilization relative to instance and allocated storage configuration. Refer to troubleshoot the latency of Amazon EBS volumes caused by an IOPS bottleneck. For Aurora, you can switch to an instance class that has I/O-Optimized storage configuration. See Planning I/O in Aurora for guidance.

Intent: This alarm is used to detect high write latency. Although database disks typically have low read/write latency, they may experience problems that cause high latency operations. Monitoring this will assure you the disk latency is as low as expected.

Statistic: p90

Recommended threshold: Depends on your situation

Threshold justification: The recommended threshold value for this alarm is highly dependent on your use case. Write latencies higher than 20 milliseconds are likely a cause for investigation. You can also set a higher threshold if your application can have a higher latency for write operations. Review the criticality and requirements of write latency and analyze the historical behavior of this metric to determine sensible threshold levels.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

DBLoad

Dimensions: DBInstanceIdentifier

Alarm description: This alarm helps to monitor high DB load. If the number of processes exceed the number of vCPUs, the processes start queuing. When the queuing increases, the performance is impacted. If the DB load is often above the maximum vCPU, and the primary wait state is CPU, the CPU is overloaded. In this case, you can monitor `CPUUtilization`, `DBLoadCPU` and queued tasks in Performance Insights/Enhanced Monitoring. You might want to throttle connections to the instance, tune any SQL queries with a high CPU load, or consider a larger instance class. High and consistent instances of any wait state indicate that there might be bottlenecks or resource contention issues to resolve.

Intent: This alarm is used to detect a high DB load. High DB load can cause performance issues in the DB instance. This alarm is not applicable to serverless DB instances.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: The maximum vCPU value is determined by the number of vCPU (virtual CPU) cores for your DB instance. Depending on the maximum vCPU, different values for the threshold can be appropriate. Ideally, DB load should not go above vCPU line.

Period: 60

Datapoints to alarm: 15

Evaluation periods: 15

Comparison Operator: GREATER_THAN_THRESHOLD

AuroraVolumeBytesLeftTotal

Dimensions: DBClusterIdentifier

Alarm description: This alarm helps to monitor low remaining total volume. When the total volume left reaches the size limit, the cluster reports an out-of-space error. Aurora storage automatically scales with the data in the cluster volume and expands up to 128 TiB or 64 TiB depending on the DB engine version. Consider reducing storage by dropping tables and databases that you no longer need. For more information, check storage scaling.

Intent: This alarm is used to detect how close the Aurora cluster is to the volume size limit. This alarm can prevent an out-of-space error that occurs when your cluster runs out of space. This alarm is recommended only for Aurora MySQL.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: You should calculate 10%-20% of the actual size limit based on velocity and trend of volume usage increase, and then use that result as the threshold value to proactively take action before the volume reaches its limit.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: LESS_THAN_THRESHOLD

AuroraBinlogReplicaLag

Dimensions: DBClusterIdentifier, Role=WRITER

Alarm description: This alarm helps to monitor the error state of Aurora writer instance replication. For more information, see [Replicating Aurora MySQL DB clusters across Amazon Regions]. For troubleshooting, see [Aurora MySQL replication issues].

Intent: This alarm is used to detect whether the writer instance is in an error state and can't replicate the source. This alarm is recommended only for Aurora MySQL.

Statistic: Average

Recommended threshold: -1.0

Threshold justification: We recommend that you use -1 as the threshold value because Aurora MySQL publishes this value if the replica is in an error state.

Period: 60

Datapoints to alarm: 2

Evaluation periods: 2

Comparison Operator: LESS_THAN_OR_EQUAL_TO_THRESHOLD

BlockedTransactions

Dimensions: DBInstanceIdentifier

Alarm description: This alarm helps to monitor a high blocked transaction count in an Aurora DB instance. Blocked transactions can end in either a rollback or a commit. High concurrency, idles in transaction, or long running transactions can lead to blocked transactions. For troubleshooting, see Aurora MySQL documentation.

Intent: This alarm is used to detect a high count of blocked transactions in an Aurora DB instance in order to prevent transaction rollbacks and performance degradation.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: You should calculate 5% of all transactions of your instance using the `ActiveTransactions` metric and use that result as the threshold value. You can also review the criticality and requirements of blocked transactions and analyze the historical behavior of this metric to determine sensible threshold levels.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

BufferCacheHitRatio

Dimensions: DBInstanceIdentifier

Alarm description: This alarm helps you monitor a consistent low cache hit ratio of the Aurora cluster. A low hit ratio indicates that your queries on this DB instance are frequently going to disk. For troubleshooting, investigate your workload to see which queries are causing this behavior, and see the DB instance RAM recommendations document.

Intent: This alarm is used to detect consistent low cache hit ratio in order to prevent a sustained performance decrease in the Aurora instance.

Statistic: Average

Recommended threshold: 80.0

Threshold justification: You can set the threshold for buffer cache hit ratio to 80%. However, you can adjust this value based on your acceptable performance level and workload characteristics.

Period: 60

Datapoints to alarm: 10

Evaluation periods: 10

Comparison Operator: LESS_THAN_THRESHOLD

EngineUptime

Dimensions: DBClusterIdentifier, Role=WRITER

Alarm description: This alarm helps to monitor low downtime of the writer DB instance. The writer DB instance can go down due to a reboot, maintenance, upgrade, or failover. When the uptime reaches 0 because of a failover in the cluster, and the cluster has one or more Aurora Replicas, then an Aurora Replica is promoted to the primary writer instance during a failure event. To increase the availability of your DB cluster, consider creating one or more Aurora Replicas in two or more different Availability Zones. For more information check factors that influence Aurora downtime.

Intent: This alarm is used to detect whether the Aurora writer DB instance is in downtime. This can prevent long-running failure in the writer instance that occurs because of a crash or failover.

Statistic: Average

Recommended threshold: 0.0

Threshold justification: A failure event results in a brief interruption, during which read and write operations fail with an exception. However, service is typically restored in less than 60 seconds, and often less than 30 seconds.

Period: 60

Datapoints to alarm: 2

Evaluation periods: 2

Comparison Operator: LESS_THAN_OR_EQUAL_TO_THRESHOLD

RollbackSegmentHistoryListLength

Dimensions: DBInstanceIdentifier

Alarm description: This alarm helps to monitor a consistent high rollback segment history length of an Aurora instance. A high InnoDB history list length indicates that a large number of old row versions, queries and database shutdowns have become slower. For more information and troubleshooting, see the InnoDB history list length increased significantly documentation.

Intent: This alarm is used to detect consistent high rollback segment history length. This can help you prevent sustained performance degradation and high CPU usage in the Aurora instance. This alarm is recommended only for Aurora MySQL.

Statistic: Average

Recommended threshold: 1000000.0

Threshold justification: Setting this threshold to 1 million should give you time to investigate the problem. However, you can adjust this value based on your acceptable performance level and workload characteristics.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

StorageNetworkThroughput

Dimensions: DBClusterIdentifier, Role=WRITER

Alarm description: This alarm helps to monitor high storage network throughput. If storage network throughput passes the total network bandwidth of the EC2 instance, it can lead to high read and write latency, which can cause degraded performance. You can check your EC2 instance type from Amazon Console. For troubleshooting, check any changes on write/read latencies and evaluate if you've also hit an alarm on this metric. If that is the case, evaluate your workload pattern during the times that the alarm was triggered. This can help you identify if you can optimize y our workload to reduce the total amount of network traffic. If this is not possible, you might need to consider scaling your instance.

Intent: This alarm is used to detect high storage network throughput. Detecting high throughput can prevent network packet drops and degraded performance.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: You should calculate about 80%-90% of the total network bandwidth of the EC2 instance type, and then use that result as the threshold value to proactively take action before the network packets are affected. You can also review the criticality and requirements of storage network throughput and analyze the historical behavior of this metric to determine sensible threshold levels.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

Amazon Route 53 Public Data Plane

HealthCheckStatus

Dimensions: HealthCheckId

Alarm description: This alarm helps to detect unhealthy endpoints as per health checkers. To understand the reason for a failure that results in unhealthy status, use the Health Checkers tab in the Route 53 Health Check Console to view the status from each Region as well as the last failure of the health check. The status tab also displays the reason that the endpoint is reported as unhealthy. Refer to troubleshooting steps.

Intent: This alarm uses Route53 health checkers to detect unhealthy endpoints.

Statistic: Average

Recommended threshold: 1.0

Threshold justification: The status of the endpoint is reported as 1 when it's healthy. Everything less than 1 is unhealthy.

Period: 60

Datapoints to alarm: 3

Evaluation periods: 3

Comparison Operator: LESS_THAN_THRESHOLD

Amazon S3

4xxErrors

Dimensions: BucketName, FilterId

Alarm description: This alarm helps us report the total number of 4xx error status codes that are made in response to client requests. 403 error codes might indicate an incorrect IAM policy, and 404 error codes might indicate mis-behaving client application, for example. Enabling S3 server access logging on a temporary basis will help you to pinpoint the issue's origin using the fields HTTP status and Error Code. To understand more about the error code, see Error Responses.

Intent: This alarm is used to create a baseline for typical 4xx error rates so that you can look into any abnormalities that might indicate a setup issue.

Statistic: Average

Recommended threshold: 0.05

Threshold justification: The recommended threshold is to detect if more than 5% of total requests are getting 4XX errors. Frequently occurring 4XX errors should be alarmed. However, setting a very low value for the threshold can cause alarm to be too sensitive. You can also tune the threshold to suit to the load of the requests, accounting for an acceptable level of 4XX errors. You can also analyze historical data to find the acceptable error rate for the application workload, and then tune the threshold accordingly.

Period: 60

Datapoints to alarm: 15

Evaluation periods: 15

Comparison Operator: GREATER_THAN_THRESHOLD

5xxErrors

Dimensions: BucketName, FilterId

Alarm description: This alarm helps you detect a high number of server-side errors. These errors indicate that a client made a request that the server couldn't complete. This can help you correlate the issue your application is facing because of S3. For more information to help you efficiently handle or reduce errors, see Optimizing performance design patterns. Errors might also be caused by an the issue with S3, check Amazon service health dashboard for the status of Amazon S3 in your Region.

Intent: This alarm can help to detect if the application is experiencing issues due to 5xx errors.

Statistic: Average

Recommended threshold: 0.05

Threshold justification: We recommend setting the threshold to detect if more than 5% of total requests are getting 5XXError. However, you can tune the threshold to suit the traffic of the requests, as well as acceptable error rates. You can also analyze historical data to see what is the acceptable error rate for the application workload, and tune the threshold accordingly.

Period: 60

Datapoints to alarm: 15

Evaluation periods: 15

Comparison Operator: GREATER_THAN_THRESHOLD

OperationsFailedReplication

Dimensions: SourceBucket, DestinationBucket, RuleId

Alarm description: This alarm helps in understanding a replication failure. This metric tracks the status of new objects replicated using S3 CRR or S3 SRR, and also tracks existing objects replicated using S3 batch replication. See Replication troubleshooting for more details.

Intent: This alarm is used to detect if there is a failed replication operation.

Statistic: Maximum

Recommended threshold: 0.0

Threshold justification: This metric emits a value of 0 for successful operations, and nothing when there are no replication operations carried out for the minute. When the metric emits a value greater than 0, the replication operation is unsuccessful.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

`S3ObjectLambda`

4xxErrors

Dimensions: AccessPointName, DataSourceARN

Alarm description: This alarm helps us report the total number of 4xx error status code that are made in response to client requests. Enabling S3 server access logging on a temporary basis will help you to pinpoint the issue's origin using the fields HTTP status and Error Code.

Intent: This alarm is used to create a baseline for typical 4xx error rates so that you can look into any abnormalities that might indicate a setup issue.

Statistic: Average

Recommended threshold: 0.05

Threshold justification: We recommend setting the threshold to detect if more than 5% of total requests are getting 4XXError. Frequently occurring 4XX errors should be alarmed. However, setting a very low value for the threshold can cause alarm to be too sensitive. You can also tune the threshold to suit to the load of the requests, accounting for an acceptable level of 4XX errors. You can also analyze historical data to find the acceptable error rate for the application workload, and then tune the threshold accordingly.

Period: 60

Datapoints to alarm: 15

Evaluation periods: 15

Comparison Operator: GREATER_THAN_THRESHOLD

5xxErrors

Dimensions: AccessPointName, DataSourceARN

Alarm description: This alarm helps to detect high number of server-side errors. These errors indicate that a client made a request that the server couldn’t complete. These errors might be caused by an issue with S3, check Amazon service health dashboard for the status of Amazon S3 in your Region. This can help you correlate the issue your application is facing because of S3. For information to help you efficiently handle or reduce these errors, see Optimizing performance design patterns.

Intent: This alarm can help to detect if the application is experiencing issues due to 5xx errors.

Statistic: Average

Recommended threshold: 0.05

Threshold justification: We recommend setting the threshold to detect if more than 5% of total requests are getting 5XX errors. However, you can tune the threshold to suit the traffic of the requests, as well as acceptable error rates. You can also analyze historical data to see what is the acceptable error rate for the application workload, and tune the threshold accordingly.

Period: 60

Datapoints to alarm: 15

Evaluation periods: 15

Comparison Operator: GREATER_THAN_THRESHOLD

LambdaResponse4xx

Dimensions: AccessPointName, DataSourceARN

Alarm description: This alarm helps you detect and diagnose failures (500s) in calls to S3 Object Lambda. These errors can be caused by errors or misconfigurations in the Lambda function responsible for responding to your requests. Investigating the CloudWatch Log Streams of the Lambda function associated with the Object Lambda Access Point can help you pinpoint the issue's origin based on the response from S3 Object Lambda.

Intent: This alarm is used to detect 4xx client errors for WriteGetObjectResponse calls.

Statistic: Average

Recommended threshold: 0.05

Threshold justification: We recommend setting the threshold to detect if more than 5% of total requests are getting 4XXError. Frequently occurring 4XX errors should be alarmed. However, setting a very low value for the threshold can cause alarm to be too sensitive. You can also tune the threshold to suit to the load of the requests, accounting for an acceptable level of 4XX errors. You can also analyze historical data to find the acceptable error rate for the application workload, and then tune the threshold accordingly.

Period: 60

Datapoints to alarm: 15

Evaluation periods: 15

Comparison Operator: GREATER_THAN_THRESHOLD

Amazon SNS

NumberOfMessagesPublished

Dimensions: TopicName

Alarm description: This alarm can detect when the number of SNS messages published is too low. For troubleshooting, check why the publishers are sending less traffic.

Intent: This alarm helps you proactively monitor and detect significant drops in notification publishing. This helps you identify potential issues with your application or business processes, so that you can take appropriate actions to maintain the expected flow of notifications. You should create this alarm if you expect your system to have a minimum traffic that it is serving.

Statistic: Sum

Recommended threshold: Depends on your situation

Threshold justification: The number of messages published should be in line with the expected number of published messages for your application. You can also analyze the historical data, trends and traffic to find the right threshold.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: LESS_THAN_THRESHOLD

NumberOfNotificationsDelivered

Dimensions: TopicName

Alarm description: This alarm can detect when the number of SNS messages delivered is too low. This could be because of unintentional unsubscribing of an endpoint, or because of an SNS event that causes messages to experience delay.

Intent: This alarm helps you detect a drop in the volume of messages delivered. You should create this alarm if you expect your system to have a minimum traffic that it is serving.

Statistic: Sum

Recommended threshold: Depends on your situation

Threshold justification: The number of messages delivered should be in line with the expected number of messages produced and the number of consumers. You can also analyze the historical data, trends and traffic to find the right threshold.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: LESS_THAN_THRESHOLD

NumberOfNotificationsFailed

Dimensions: TopicName

Alarm description: This alarm can detect when the number of failed SNS messages is too high. To troubleshoot failed notifications, enable logging to CloudWatch Logs. Checking the logs can help you find which subscribers are failing, as well as the status codes they are returning.

Intent: This alarm helps you proactively find issues with the delivery of notifications and take appropriate actions to address them.

Statistic: Sum

Recommended threshold: Depends on your situation

Threshold justification: The recommended threshold value for this alarm is highly dependent on the impact of failed notifications. Review the SLAs provided to your end users, fault tolerance and criticality of notifications and analyze historical data, and then select a threshold accordingly. The number of notifications failed should be 0 for topics that have only SQS, Lambda or Firehose subscriptions.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

NumberOfNotificationsFilteredOut-InvalidAttributes

Dimensions: TopicName

Alarm description: This alarm helps to monitor and resolve potential problems with the publisher or subscribers. Check if a publisher is publishing messages with invalid attributes or if an inappropriate filter is applied to a subscriber. You can also analyze CloudWatch Logs to help find the root cause of the issue.

Intent: The alarm is used to detect if the published messages are not valid or if inappropriate filters have been applied to a subscriber.

Statistic: Sum

Recommended threshold: 0.0

Threshold justification: Invalid attributes are almost always a mistake by the publisher. We recommend to set the threshold to 0 because invalid attributes are not expected in a healthy system.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

NumberOfNotificationsFilteredOut-InvalidMessageBody

Dimensions: TopicName

Alarm description: This alarm helps to monitor and resolve potential problems with the publisher or subscribers. Check if a publisher is publishing messages with invalid message bodies, or if an inappropriate filter is applied to a subscriber. You can also analyze CloudWatch Logs to help find the root cause of the issue.

Intent: The alarm is used to detect if the published messages are not valid or if inappropriate filters have been applied to a subscriber.

Statistic: Sum

Recommended threshold: 0.0

Threshold justification: Invalid message bodies are almost always a mistake by the publisher. We recommend to set the threshold to 0 because invalid message bodies are not expected in a healthy system.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

NumberOfNotificationsRedrivenToDlq

Dimensions: TopicName

Alarm description: This alarm helps to monitor the number of messages that are moved to a dead-letter queue.

Intent: The alarm is used to detect messages that moved to a dead-letter queue. We recommend that you create this alarm when SNS is coupled with SQS, Lambda or Firehose.

Statistic: Sum

Recommended threshold: 0.0

Threshold justification: In a healthy system of any subscriber type, messages should not be moved to the dead-letter queue. We recommend that you be notified if any messages land in the queue, so that you can identify and address the root cause, and potentially redrive the messages in the dead-letter queue to prevent data loss.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

NumberOfNotificationsFailedToRedriveToDlq

Dimensions: TopicName

Alarm description: This alarm helps to monitor messages that couldn't be moved to a dead-letter queue. Check whether your dead-letter queue exists and that it's configured correctly. Also, verify that SNS has permissions to access the dead-letter queue. Refer to the dead-letter queue documentation to learn more.

Intent: The alarm is used to detect messages that couldn't be moved to a dead-letter queue.

Statistic: Sum

Recommended threshold: 0.0

Threshold justification: It's almost always a mistake if messages can't be moved to the dead-letter queue. The recommendation for the threshold is 0, meaning all messages that fail processing must be able to reach the dead-letter queue when the queue has been configured.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

SMSMonthToDateSpentUSD

Dimensions: TopicName

Alarm description: The alarm helps to monitor if you have a sufficient quota in your account for SNS to be able to deliver messages. If you reach your quota, SNS won't be able to deliver SMS messages. For information about setting your monthly SMS spend quota, or for information about requesting a spend quota increase with Amazon, see Setting SMS messaging preferences.

Intent: This alarm is used to detect if you have a sufficient quota in your account for your SMS messages to be delivered successfully.

Statistic: Maximum

Recommended threshold: Depends on your situation

Threshold justification: Set the threshold in accordance with the quota (Account spend limit) for the account. Choose a threshold which informs you early enough that you are reaching your quota limit so that you have time to request an increase.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

SMSSuccessRate

Dimensions: TopicName

Alarm description: This alarm helps to monitor the rate of failing SMS message deliveries. You can set up Cloudwatch Logs to understand the nature of the failure and take action based on that.

Intent: This alarm is used to detect failing SMS message deliveries.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: Set the threshold for the alarm in line with your tolerance for failing SMS message deliveries.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

Amazon SQS

ApproximateAgeOfOldestMessage

Dimensions: QueueName

Alarm description: This alarm watches the age of the oldest message in the queue. You can use this alarm to monitor if your consumers are processing SQS messages at the desired speed. Consider increasing the consumer count or consumer throughput to reduce message age. This metric can be used in combination with `ApproximateNumberOfMessagesVisible` to determine how big the queue backlog is and how quickly messages are being processed. To prevent messages from being deleted before processed, consider configuring the dead-letter queue to sideline potential poison pill messages.

Intent: This alarm is used to detect whether the age of the oldest message in the QueueName queue is too high. High age can be an indication that messages are not processed quickly enough or that there are some poison-pill messages that are stuck in the queue and can't be processed.

Statistic: Maximum

Recommended threshold: Depends on your situation

Threshold justification: The recommended threshold value for this alarm is highly dependent on the expected message processing time. You can use historical data to calculate the average message processing time, and then set the threshold to 50% higher than the maximum expected SQS message processing time by queue consumers.

Period: 60

Datapoints to alarm: 15

Evaluation periods: 15

Comparison Operator: GREATER_THAN_OR_EQUAL_TO_THRESHOLD

ApproximateNumberOfMessagesNotVisible

Dimensions: QueueName

Alarm description: This alarm helps to detect a high number of in-flight messages with respect to `QueueName`. For troubleshooting, check [message backlog decreasing](#).

Intent: This alarm is used to detect a high number of in-flight messages in the queue. If consumers do not delete messages within the visibility timeout period, when the queue is polled, messages reappear in the queue. For FIFO queues, there can be a maximum of 20,000 in-flight messages. If you reach this quota, SQS returns no error messages. A FIFO queue looks through the first 20k messages to determine available message groups. This means that if you have a backlog of messages in a single message group, you cannot consume messages from other message groups that were sent to the queue at a later time until you successfully consume the messages from the backlog.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: The recommended threshold value for this alarm is highly dependent on the expected number of messages in flight. You can use historical data to calculate the maximum expected number of messages in flight and set the threshold to 50% over this value. If consumers of the queue are processing but not deleting messages from the queue, this number will suddenly increase.

Period: 60

Datapoints to alarm: 15

Evaluation periods: 15

Comparison Operator: GREATER_THAN_OR_EQUAL_TO_THRESHOLD

ApproximateNumberOfMessagesVisible

Dimensions: QueueName

Alarm description: This alarm watches for the message queue backlog to be bigger than expected, indicating that consumers are too slow or there are not enough consumers. Consider increasing the consumer count or speeding up consumers, if this alarm goes into ALARM state.

Intent: This alarm is used to detect whether the message count of the active queue is too high and consumers are slow to process the messages or there are not enough consumers to process them.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: An unexpectedly high number of messages visible indicates that messages are not being processed by a consumer at the expected rate. You should consider historical data when you set this threshold.

Period: 60

Datapoints to alarm: 15

Evaluation periods: 15

Comparison Operator: GREATER_THAN_OR_EQUAL_TO_THRESHOLD

NumberOfMessagesSent

Dimensions: QueueName

Alarm description: This alarm helps to detect if there are no messages being sent from a producer with respect to `QueueName`. For troubleshooting, check the reason that the producer is not sending messages.

Intent: This alarm is used to detect when a producer stops sending messages.

Statistic: Sum

Recommended threshold: 0.0

Threshold justification: If the number of messages sent is 0, the producer is not sending any messages. If this queue has a low TPS, increase the number of EvaluationPeriods accordingly.

Period: 60

Datapoints to alarm: 15

Evaluation periods: 15

Comparison Operator: LESS_THAN_OR_EQUAL_TO_THRESHOLD

Amazon VPN

TunnelState

Dimensions: VpnId

Alarm description: This alarm helps you understand if the state of one or more tunnels is DOWN. For troubleshooting, see [VPN tunnel troubleshooting].

Intent: This alarm is used to detect if at least one tunnel is in the DOWN state for this VPN, so that you can troubleshoot the impacted VPN. This alarm will always be in the ALARM state for networks that only have a single tunnel configured.

Statistic: Minimum

Recommended threshold: 1.0

Threshold justification: A value less than 1 indicates that at least one tunnel is in DOWN state.

Period: 300

Datapoints to alarm: 3

Evaluation periods: 3

Comparison Operator: LESS_THAN_THRESHOLD

TunnelState

Dimensions: TunnelIpAddress

Alarm description: This alarm helps you understand if the state of this tunnel is DOWN. For troubleshooting, see [VPN tunnel troubleshooting].

Intent: This alarm is used to detect if the tunnel is in the DOWN state, so that you can troubleshoot the impacted VPN. This alarm will always be in the ALARM state for networks that only have a single tunnel configured.

Statistic: Minimum

Recommended threshold: 1.0

Threshold justification: A value less than 1 indicates that the tunnel is in DOWN state.

Period: 300

Datapoints to alarm: 3

Evaluation periods: 3

Comparison Operator: LESS_THAN_THRESHOLD

Alarming on metrics

The steps in the following sections explain how to create CloudWatch alarms on metrics.

Create a CloudWatch alarm based on a static threshold

You choose a CloudWatch metric for the alarm to watch, and the threshold for that metric. The alarm goes to `ALARM` state when the metric breaches the threshold for a specified number of evaluation periods.

If you are creating an alarm in an account set up as a monitoring account in CloudWatch cross-account observability, you can set up the alarm to watch a metric in a source account linked to this monitoring account. For more information, see [CloudWatch cross-account observability](#).

To create an alarm based on a single metric

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Alarms**, **All alarms**.
3. Choose **Create alarm**.
4. Choose **Select Metric**.
5. Do one of the following:
   - Choose the service namespace that contains the metric that you want. Continue choosing options as they appear to narrow the choices. When a list of metrics appears, select the check box next to the metric that you want.
   - In the search box, enter the name of a metric, account ID, account label, dimension, or resource ID. Then, choose one of the results and continue until a list of metrics appears. Select the check box next to the metric that you want.
6. Choose the **Graphed metrics** tab.

   a. Under **Statistic** , choose one of the statistics or predefined percentiles, or specify a custom percentile (for example, **`p95.45`**).

   b. Under **Period**, choose the evaluation period for the alarm. When evaluating the alarm, each period is aggregated into one data point.

      You can also choose whether the y-axis legend appears on the left or right while you're creating the alarm. This preference is used only while you're creating the alarm.

   c. Choose **Select metric**.

The **Specify metric and conditions** page appears, showing a graph and other information about the metric and statistic that you selected.

7. Under **Conditions**, specify the following:

   a. For **Whenever *`metric`* is**, specify whether the metric must be greater than, less than, or equal to the threshold. Under **than...**, specify the threshold value.

   b. Choose **Additional configuration**. For **Datapoints to alarm**, specify how many evaluation periods (data points) must be in the `ALARM` state to trigger the alarm. If the two values here match, you create an alarm that goes to `ALARM` state if that many consecutive periods are breaching.

      To create an M out of N alarm, specify a lower number for the first value than you specify for the second value. For more information, see [Evaluating an alarm](#).

   c. For **Missing data treatment**, choose how to have the alarm behave when some data points are missing. For more information, see [Configuring how CloudWatch alarms treat missing data](#).

   d. If the alarm uses a percentile as the monitored statistic, a **Percentiles with low samples** box appears. Use it to choose whether to evaluate or ignore cases with low sample rates. If you choose **ignore (maintain alarm state)**, the current alarm state is always maintained when the sample size is too low. For more information, see [Percentile-based CloudWatch alarms and low data samples](#).

8. Choose **Next**.

9. Under **Notification**, select an SNS topic to notify when the alarm is in `ALARM` state, `OK` state, or `INSUFFICIENT_DATA` state.

   To have the alarm send multiple notifications for the same alarm state or for different alarm states, choose **Add notification**.

   In CloudWatch cross-account observability, you can choose to have notifications sent to multiple Amazon accounts. For example, to both the monitoring account and the source account.

   To have the alarm not send notifications, choose **Remove**.

10. To have the alarm perform Auto Scaling, EC2, Lambda, investigation, or Systems Manager actions, choose the appropriate button and choose the alarm state and action to perform. Alarms can perform Systems Manager actions and investigation actions only when they go

into ALARM state. For more information about Systems Manager actions, see Configuring CloudWatch to create OpsItems from alarms and Incident creation.

To have the alarm start an investigation, choose **Add investigation action** and then select your investigation group. For more information about Amazon Q Developer operational investigations, see Amazon Q Developer operational investigations (Preview).

> **Note**
>
> To create an alarm that performs an SSM Incident Manager action, you must have certain permissions. For more information, see Identity-based policy examples for Amazon Systems Manager Incident Manager.

11. When finished, choose **Next**.
12. Enter a name and description for the alarm. The name must contain only UTF-8 characters, and can't contain ASCII control characters. The description can include markdown formatting, which is displayed only in the alarm **Details** tab in the CloudWatch console. The markdown can be useful to add links to runbooks or other internal resources. Then choose **Next**.
13. Under **Preview and create**, confirm that the information and conditions are what you want, then choose **Create alarm**.

You can also add alarms to a dashboard. For more information, see Adding an alarm to a CloudWatch dashboard.

Create a CloudWatch alarm based on a metric math expression

To create an alarm based on a metric math expression, choose one or more CloudWatch metrics to use in the expression. Then, specify the expression, threshold, and evaluation periods.

You can't create an alarm based on the **SEARCH** expression. This is because search expressions return multiple time series, and an alarm based on a math expression can watch only one time series.

To create an alarm that's based on a metric math expression

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Alarms**, and then choose **All alarms**.
3. Choose **Create alarm**.

4. Choose **Select Metric**, and then perform one of the following actions:
   - Select a namespace from the **Amazon namespaces** dropdown or **Custom namespaces** dropdown. After you select a namespace, you continue choosing options until a list of metrics appears, where you select the checkbox next to the correct metric.
   - Use the search box to find a metric, account ID, dimension, or resource ID. After you enter the metric, dimension, or resource ID, you continue choosing options until a list of metrics appears, where you select the check box next to the correct metric.
5. (Optional) If you want to add another metric to a metric math expression, you can use the search box to find a specific metric. You can add as many as 10 metrics to a metric math expression.
6. Select the **Graphed metrics** tab. For each of the metrics that you previously added, perform the following actions:

   a. Under the **Statistic** column, select the dropdown menu. In the dropdown menu, choose one of the predefined statistics or percentiles. Use the search box in the dropdown menu to specify a custom percentile.

   b. Under the **Period** column, select the dropdown menu. In the dropdown menu, choose one of the predefined evaluation periods.

      While you're creating your alarm, you can specify whether the Y-axis legend appears on the left or right side of your graph.

   > **Note**
   >
   > When CloudWatch evaluates alarms, periods are aggregated into single data points.

7. Choose the **Add math** dropdown, and then select **Start with an empty expression** from the list of predefined metric math expressions.

   After you choose **Start with an empty expression**, a math expression box appears where you apply or edit math expressions.
8. In the math expression box, enter your math expression, and then choose **Apply**.

   After you choose **Apply**, an **ID** column appears next to the **Label** column.

   To use a metric or the result of another metric math expression as part of your current math expression's formula, you use the value that's shown under the **ID** column. To change the value of **ID**, you select the pen-and-paper icon next to the current value. The new value must begin with a lowercase letter and can include numbers, letters, and the underscore symbol. Changing the value of **ID** to a more significant name can make your alarm graph easier to understand.

For information about the functions that are available for metric math, see [Metric math syntax and functions](#).

9. (Optional) Add more math expressions, using both metrics and the results of other math expressions in the formulas of the new math expressions.

10. When you have the expression to use for the alarm, clear the check boxes to the left of every other expression and every metric on the page. Only the check box next to the expression to use for the alarm should be selected. The expression that you choose for the alarm must produce a single time series and show only one line on the graph. Then choose **Select metric**.

    The **Specify metric and conditions** page appears, showing a graph and other information about the math expression that you have selected.

11. For **Whenever *`expression`* is**, specify whether the expression must be greater than, less than, or equal to the threshold. Under **than...**, specify the threshold value.

12. Choose **Additional configuration**. For **Datapoints to alarm**, specify how many evaluation periods (data points) must be in the `ALARM` state to trigger the alarm. If the two values here match, you create an alarm that goes to `ALARM` state if that many consecutive periods are breaching.

    To create an M out of N alarm, specify a lower number for the first value than you specify for the second value. For more information, see [Evaluating an alarm](#).

13. For **Missing data treatment**, choose how to have the alarm behave when some data points are missing. For more information, see [Configuring how CloudWatch alarms treat missing data](#).

14. Choose **Next**.

15. Under **Notification**, select an SNS topic to notify when the alarm is in `ALARM` state, `OK` state, or `INSUFFICIENT_DATA` state.

    To have the alarm send multiple notifications for the same alarm state or for different alarm states, choose **Add notification**.

    To have the alarm not send notifications, choose **Remove**.

16. To have the alarm perform Auto Scaling, EC2, Lambda, or Systems Manager actions, choose the appropriate button and choose the alarm state and action to perform. If you choose a

Lambda function as an alarm action, you specify the function name or ARN, and you can optionally choose a specific version of the function.

Alarms can perform Systems Manager actions only when they go into ALARM state. For more information about Systems Manager actions, see see Configuring CloudWatch to create OpsItems from alarms and Incident creation.

> **Note**
>
> To create an alarm that performs an SSM Incident Manager action, you must have certain permissions. For more information, see Identity-based policy examples for Amazon Systems Manager Incident Manager.

17. When finished, choose **Next**.
18. Enter a name and description for the alarm. Then choose **Next**.

    The name must contain only UTF-8 characters, and can't contain ASCII control characters. The description can include markdown formatting, which is displayed only in the alarm **Details** tab in the CloudWatch console. The markdown can be useful to add links to runbooks or other internal resources.
19. Under **Preview and create**, confirm that the information and conditions are what you want, then choose **Create alarm**.

You can also add alarms to a dashboard. For more information, see Adding an alarm to a CloudWatch dashboard.

Create a CloudWatch alarm based on a Metrics Insights query

You can create an alarm on any Metrics Insights query that returns a single time series. This can be especially useful to create dynamic alarms that watch aggregated metrics across a fleet of your infrastructure or applications. Create the alarm once, and it adjusts as resources are added to or removed from the fleet. For example, you can create an alarm that watches the CPU ulitization of all your instances, and the alarm dynamically adjusts as you add or remove instances.

For complete instructions, see Alarms on CloudWatch Metrics Insights queries in CloudWatch.

Create an alarm based on a connected data source

You can create alarms that watch metrics from data sources that aren't in CloudWatch. For more information about creating connections to these other data sources, see Query metrics from other data sources.

To create an alarm on metrics from a data source that you have connected to

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Metrics**, **All metrics**.
3. Choose the **Multi source query** tab.
4. For **Data source**, select the data source that you want to use.
5. The query builder prompts you for the information necessary for the query to retrieve the metrics to use for the alarm. The workflow is different for each data source, and is tailored to the data source. For example, for Amazon Managed Service for Prometheus and Prometheus data sources, a PromQL query editor box with a query helper appears.
6. When you have finished constructing the query, choose **Graph query**.
7. If the sample graph looks the way that you expect, choose **Create alarm**.
8. The **Specify metric and conditions** page appears. If the query you are using produces more than one time series, you see a warning banner at the top of the page. If you do, select a function to use to aggregate the time series in **Aggregation function**.
9. (Optional) Add a **Label** for the alarm.
10. For **Whenever *`your-metric-name`* is . . .**, choose **Greater**, **Greater/Equal**, **Lower/Equal**, or **Lower**. Then for **than . . .**, specify a number for your threshold value.
11. Choose **Additional configuration**. For **Datapoints to alarm**, specify how many evaluation periods (data points) must be in the `ALARM` state to trigger the alarm. If the two values here match, you create an alarm that goes to `ALARM` state if that many consecutive periods are breaching.

    To create an M out of N alarm, specify a number for the first value that is lower than the number for the second value. For more information, see Evaluating an alarm.
12. For **Missing data treatment**, choose how the alarm behaves when some data points are missing. For more information, see Configuring how CloudWatch alarms treat missing data.
13. Choose **Next**.

14. For **Notification**, specify an Amazon SNS topic to notify when your alarm transitions to the `ALARM`, `OK`, or `INSUFFICIENT_DATA` state.

    a. (Optional) To send multiple notifications for the same alarm state or for different alarm states, choose **Add notification**.

    > **Note**
    >
    > We recommend that you set the alarm to take actions when it goes into **Insufficient data** state in addition to when it goes into **Alarm** state. This is because many issues with the Lambda function that connects to the data source can cause the alarm to transition to **Insufficient data**.

    b. (Optional) To not send Amazon SNS notifications, choose **Remove**.

15. To have the alarm perform Auto Scaling, Lambda, or Systems Manager actions, choose the appropriate button and choose the alarm state and action to perform. If you choose a Lambda function as an alarm action, you specify the function name or ARN, and you can optionally choose a specific version of the function.

    Alarms can perform Systems Manager actions only when they go into ALARM state. For more information about Systems Manager actions, see see Configuring CloudWatch to create OpsItems from alarms and Incident creation.

    > **Note**
    >
    > To create an alarm that performs an SSM Incident Manager action, you must have certain permissions. For more information, see Identity-based policy examples for Amazon Systems Manager Incident Manager.

16. Choose **Next**.

17. Under **Name and description**, enter a name and description for your alarm, and choose **Next**. The name must contain only UTF-8 characters, and can't contain ASCII control characters. The description can include markdown formatting, which is displayed only in the alarm **Details** tab in the CloudWatch console. The markdown can be useful to add links to runbooks or other internal resources.

> **Tip**
>
> The alarm name must contain only UTF-8 characters. It can't contain ASCII control characters.

18. Under **Preview and create**, confirm that your alarm's information and conditions are correct, and choose **Create alarm**.

Details about alarms for connected data sources

- When CloudWatch evaluates an alarm, it does so every minute, even if the period for the alarm is longer than one minute. For the alarm to work, the Lambda function must be able to return a list of timestamps starting on any minute, not only on multiples of the period length. These timestamps must be spaced one period length apart.

  Therefore, if the data source queried by the Lambda can only return timestamps that are multiples of the period length, the function should "re-sample" the fetched data to match the timestamps expected by the `GetMetricData` request.

  For example, an alarm with a five-minute period is evaluated every minute using five-minute windows that shift by one minute each time. In this case:

  - For the alarm evaluation at 12:15:00, CloudWatch expects data points with timestamps of `12:00:00`, `12:05:00`, and `12:10:00`.
  - Then for the alarm evaluation at 12:16:00, CloudWatch expects data points with timestamps of `12:01:00`, `12:06:00`, and `12:11:00`.
- When CloudWatch evaluates an alarm, any data points returned by the Lambda function that don't align with the expected timestamps are dropped, and the alarm is evaluated using the remaining expected data points. For example, when the alarm is evaluated at `12:15:00` it expects data with timestamps of `12:00:00`, `12:05:00`, and `12:10:00`. If it receives data with timestamps of `12:00:00`, `12:05:00`, `12:06:00`, and `12:10:00`, the data from `12:06:00` is dropped and CloudWatch evaluates the alarm using the other timestamps.

  Then for the next evaluation at `12:16:00`, it expects data with timestamps of `12:01:00`, `12:06:00`, and `12:11:00`. If it only has the data with timestamps of `12:00:00`, `12:05:00`, and `12:10:00`, all of these data points are ignored at 12:16:00 and the alarm transitions into the state according to how you specified the alarm to treat missing data. For more information, see Evaluating an alarm.

- We recommend that you create these alarms to take actions when they transition to the `INSUFFICIENT_DATA` state, because several Lambda function failure use cases will transition the alarm to `INSUFFICIENT_DATA` regardless of the way that you set the alarm to treat missing data.
- If the Lambda function returns an error:
  - If there is a permission problem with calling the Lambda function, the alarm begins having missing data transitions according to how you specified the alarm to treat missing data when you created it.
  - Any other error coming from the Lambda function causes the alarm to transition to `INSUFFICIENT_DATA`.
- If the Lambda function returns partial data:
  - The alarm continues to be evaluated on the data points that are returned.
  - You can use the following methods to find whether an alarm on a Lambda function is currently evaluating its alarm state based on partial data:
    - In the console, choose an alarm and choose the **Details** page. If you see the message **Evaluation warning: Not evaluating all data appears on that page**, it is evaluating on partial data.
    - If you see the value `PARTIAL_DATA` in the `EvaluationState` field when you use the `describe-alarms` Amazon CLI command or the DescribeAlarms API, it is evaluating on partial data.
  - An alarm also publishes events to Amazon EventBridge when it goes into the partial data state.
- If the metric requested by the Lambda function has some delay so that the last data point is always missing, you should use a workaround. You can create an M out of N alarm or increase the evaluation period of the alarm. For more information about M out of N alarms, see Evaluating an alarm.

Create a CloudWatch alarm based on anomaly detection

You can create an alarm based on CloudWatch anomaly detection, which analyzes past metric data and creates a model of expected values. The expected values take into account the typical hourly, daily, and weekly patterns in the metric.

You set a value for the anomaly detection threshold, and CloudWatch uses this threshold with the model to determine the "normal" range of values for the metric. A higher value for the threshold produces a thicker band of "normal" values.

You can choose whether the alarm is triggered when the metric value is above the band of expected values, below the band, or either above or below the band.

You also can create anomaly detection alarms on single metrics and the outputs of metric math expressions. You can use these expressions to create graphs that visualize anomaly detection bands.

In an account set up as a monitoring account for CloudWatch cross-account observability, you can create anomaly detectors on metrics in source accounts in addition to metrics in the monitoring account.

For more information, see Using CloudWatch anomaly detection.

> **Note**
>
> If you're already using anomaly detection for visualization purposes on a metric in the Metrics console and you create an anomaly detection alarm on that same metric, then the threshold that you set for the alarm doesn't change the threshold that you already set for visualization. For more information, see Creating a graph.

To create an alarm that's based on anomaly detection

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Alarms**, **All alarms**.
3. Choose **Create alarm**.
4. Choose **Select Metric**.
5. Do one of the following:
   - Choose the service namespace that contains your metric, and then continue choosing options as they appear to narrow down your options. When a list of metrics appears, select the check box that's next to your metric.
   - In the search box, enter the name of a metric, dimension, or resource ID. Select one of the results, and then continue choosing options as they appear until a list of metrics appears. Select the check box that's next to your metric.

6. Choose **Graphed metric**.

   a. (Optional) For *Statistic*, choose the dropdown, and then select one of the predefined statistics or percentiles. You can use the search box in the dropdown to specify a custom percentile, such as **p95.45**.

   b. (Optional) For *Period*, choose the dropdown, and then select one of the predefined evaluation periods.

      > **Note**
      >
      > When CloudWatch evaluates your alarm, it aggregates the period into a single datapoint. For an anomaly detection alarm, the evaluation period must be one minute or longer.

7. Choose **Next**.

8. Under ***Conditions***, specify the following:

   a. Choose **Anomaly detection**.

      If the model for this metric and statistic already exists, CloudWatch displays a preview of the anomaly detection band in the graph at the top of the screen. After you create your alarm, it can take up to 15 minutes for the actual anomaly detection band to appear in the graph. Before that, the band that you see is an approximation of the anomaly detection band.

      > **Tip**
      >
      > To see the graph at the top of the screen in a longer time frame, choose **Edit** at the top-right of the screen.

      If the model for this metric and statistic doesn't already exist, CloudWatch generates the anomaly detection band after you finish creating your alarm. For new models, it can take up to 3 hours for the actual anomaly detection band to appear in your graph. It can take up to two weeks for the new model to train, so the anomaly detection band shows more accurate expected values.

   b. For **Whenever *metric* is**, specify when to trigger the alarm. For example, when the metric is greater than, lower than, or outside the band (in either direction).

   c. For **Anomaly detection threshold**, choose the number to use for the anomaly detection threshold. A higher number creates a thicker band of "normal" values that is more tolerant of metric changes. A lower number creates a thinner band that will go to `ALARM` state with smaller metric deviations. The number does not have to be a whole number.

   d. Choose **Additional configuration**. For **Datapoints to alarm**, specify how many evaluation periods (data points) must be in the `ALARM` state to trigger the alarm. If the two values here match, you create an alarm that goes to `ALARM` state if that many consecutive periods are breaching.

      To create an M out of N alarm, specify a number for the first value that is lower than the number for the second value. For more information, see [Evaluating an alarm](#).

   e. For **Missing data treatment**, choose how the alarm behaves when some data points are missing. For more information, see [Configuring how CloudWatch alarms treat missing data](#).

   f. If the alarm uses a percentile as the monitored statistic, a **Percentiles with low samples** box appears. Use it to choose whether to evaluate or ignore cases with low sample rates. If you choose **Ignore (maintain alarm state)**, the current alarm state is always maintained when the sample size is too low. For more information, see [Percentile-based CloudWatch alarms and low data samples](#).

9. Choose **Next**.

10. Under **Notification**, select an SNS topic to notify when the alarm is in `ALARM` state, `OK` state, or `INSUFFICIENT_DATA` state.

    To send multiple notifications for the same alarm state or for different alarm states, choose **Add notification**.

    Choose **Remove** if you don't want the alarm to send notifications.

11. You can set up the alarm to perform EC2 actions or invoke a Lambda function when it changes state, or to create a Systems Manager OpsItem or incident when it goes into ALARM state. To do this, choose the appropriate button and then choose the alarm state and action to perform.

    If you choose a Lambda function as an alarm action, you specify the function name or ARN, and you can optionally choose a specific version of the function.

    For more information about Systems Manager actions, see [Configuring CloudWatch to create OpsItems from alarms](#) and [Incident creation](#).

> **Note**
>
> To create an alarm that performs an Amazon Systems Manager Incident Manager action, you must have certain permissions. For more information, see [Identity-based policy examples for Amazon Systems Manager Incident Manager](#).

12. Choose **Next**.
13. Under ***Name and description***, enter a name and description for your alarm, and choose **Next**. The name must contain only UTF-8 characters, and can't contain ASCII control characters. The description can include markdown formatting, which is displayed only in the alarm **Details** tab in the CloudWatch console. The markdown can be useful to add links to runbooks or other internal resources.

> **Tip**
>
> The alarm name must contain only UTF-8 characters, and can't contain ASCII control characters

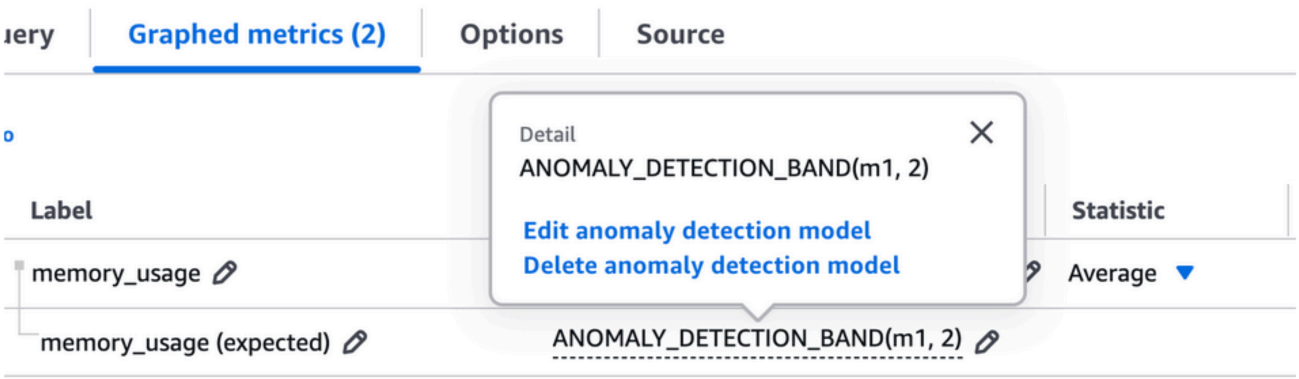
14. Under ***Preview and create***, confirm that your alarm's information and conditions are correct, and choose **Create alarm**.

Editing an anomaly detection model

After you create an alarm, you can adjust the anomaly detection model. You can exclude certain time periods from being used in the model creation. It is critical that you exclude unusual events such as system outages, deployments, and holidays from the training data. You can also specify whether to adjust the model for Daylight Savings Time changes.

To edit the anomaly detection model for an alarm

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Alarms**, **All alarms**.
3. Choose the name of the alarm. If necessary, use the search box to find the alarm.
4. Choose **View**, **In metrics**.
5. In the **Details** column, choose the **ANOMALY_DETECTION_BAND** keyword, and then choose **Edit anomaly detection model** in the popup.



6. To exclude a time period from being used to produce the model, choose the calendar icon by **End date**. Then, select or enter the days and times to exclude from training and choose **Apply**.
7. If the metric is sensitive to Daylight Savings Time changes, select the appropriate time zone in the **Metric timezone** box.
8. Choose **Update**.

Deleting an anomaly detection model

Using anomaly detection for an alarm accrues charges. As a best practice, if your alarm no longer needs an anomaly detection model, delete the alarm first and the model second. When anomaly detection alarms are evaluated, any missing anomaly detectors are created on your behalf. If you delete the model without deleting the alarm, the alarm automatically recreates the model.

To delete an alarm

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Alarms**, **All Alarms**.
3. Choose the name of the alarm.
4. Choose **Actions**, **Delete**.
5. In the confirmation box, choose **Delete**.

To delete an anomaly detection model that was used for an alarm

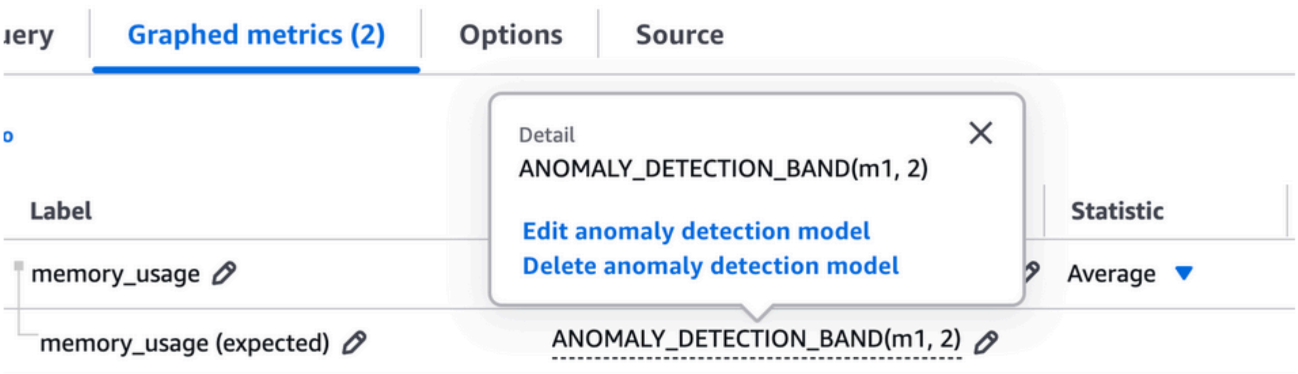
1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Metrics**, and then choose **All metrics**.

3. Choose **Browse**, and then select the metric that includes the anomaly detection model. You can search for your metric in the search box or select your metric by choosing through the options.

   - (Optional) If you're using the original interface, choose **All metrics**, and then choose the metric that includes the anomaly detection model. You can search for your metric in the search box or select your metric by choosing through the options.

4. Choose **Graphed metrics**.

5. In the **Graphed metrics** tab, in the **Details** column, choose the **ANOMALY_DETECTION_BAND** keyword, and then choose **Delete anomaly detection model** in the popup.



   - (Optional) If you're using the original interface, choose **Edit model**. You're directed to a new screen. On the new screen, choose **Delete model**, and then choose **Delete**.

Alarming on logs

The steps in the following sections explain how to create CloudWatch alarms on logs.

Create a CloudWatch alarm based on a log group-metric filter

The procedure in this section describes how to create an alarm based on a log group-metric filter. With metric filters, you can look for terms and patterns in log data as the data is sent to CloudWatch. For more information, see Create metrics from log events using filters in the *Amazon CloudWatch Logs User Guide*. Before you create an alarm based on a log group-metric filter, you must complete the following actions:

- Create a log group. For more information, see Working with log groups and log streams in the *Amazon CloudWatch Logs User Guide*.

- Create a metric filter. For more information, see [Create a metric filter for a log group]() in the *Amazon CloudWatch Logs User Guide*.

To create an alarm based on a log group-metric filter

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. From the navigation pane, choose **Logs**, and then choose **Log groups**.
3. Choose the log group that includes your metric filter.
4. Choose **Metric filters**.
5. In the metric filters tab, select the box for the metric filter that you want to base your alarm on.
6. Choose **Create alarm**.
7. (Optional) Under **Metric**, edit **Metric name**, **Statistic**, and **Period**.
8. Under **Conditions**, specify the following:

   a. For **Threshold type**, choose **Static** or **Anomaly detection**.

   b. For **Whenever *`your-metric-name`* is . . .**, choose **Greater**, **Greater/Equal**, **Lower/Equal** , or **Lower**.

   c. For **than . . .**, specify a number for your threshold value.
9. Choose **Additional configuration**.

   a. For **Data points to alarm**, specify how many data points trigger your alarm to go into the `ALARM` state. If you specify matching values, your alarm goes into the `ALARM` state if that many consecutive periods are breaching. To create an M-out-of-N alarm, specify a number for the first value that's lower than the number you specify for the second value. For more information, see Using Amazon CloudWatch alarms.

   b. For **Missing data treatment**, select an option to specify how to treat missing data when your alarm is evaluated.
10. Choose **Next**.
11. For **Notification**, specify an Amazon SNS topic to notify when your alarm is in the `ALARM`, `OK`, or `INSUFFICIENT_DATA` state.

    a. (Optional) To send multiple notifications for the same alarm state or for different alarm states, choose **Add notification**.

    b. (Optional) To not send notifications, choose **Remove**.

12. To have the alarm perform Auto Scaling, EC2, Lambda, or Systems Manager actions, choose the appropriate button and choose the alarm state and action to perform. If you choose a Lambda function as an alarm action, you specify the function name or ARN, and you can optionally choose a specific version of the function.

    Alarms can perform Systems Manager actions only when they go into ALARM state. For more information about Systems Manager actions, see see [Configuring CloudWatch to create OpsItems from alarms] and [Incident creation].

    > **Note**
    >
    > To create an alarm that performs an SSM Incident Manager action, you must have certain permissions. For more information, see [Identity-based policy examples for Amazon Systems Manager Incident Manager].

13. Choose **Next**.
14. For **Name and description**, enter a name and description for your alarm. The name must contain only UTF-8 characters, and can't contain ASCII control characters. The description can include markdown formatting, which is displayed only in the alarm **Details** tab in the CloudWatch console. The markdown can be useful to add links to runbooks or other internal resources.
15. For **Preview and create**, check that your configuration is correct, and choose **Create alarm**.

Combining alarms

With CloudWatch, you can combine several alarms into one *composite alarm* to create a summarized, aggregated health indicator over a whole application or group of resources. Composite alarms are alarms that determine their state by monitoring the states of other alarms. You define rules to combine the status of those monitored alarms using Boolean logic.

You can use composite alarms to reduce alarm noise by taking actions only at an aggregated level. For example, you can create a composite alarm to send a notification to your web server team if any alarm related to your web server triggers. When any of those alarms goes into the ALARM state, the composite alarm goes itself in the ALARM state and sends a notification to your team. If other alarms related to your web server also go into the ALARM state, your team does not get overloaded with new notifications since the composite alarm has already notified them about the existing situation.

You can also use composite alarms to create complex alarming conditions and take actions only when many different conditions are met. For example, you can create a composite alarm that combines a CPU alarm and a memory alarm, and would only notify your team if both the CPU and the memory alarms have triggered.

Using composite alarms

When you use composite alarms, you have two options:

- Configure the actions you want to take only at the composite alarm level, and create the underlying monitored alarms without actions
- Configure a different set of actions at the composite alarm level. For example, the composite alarm actions could engage a different team in case of a widespread issue.

Composite alarms can take only the following actions:

- Notify Amazon SNS topics
- Invoke Lambda functions
- Create OpsItems in Systems Manager Ops Center
- Create incidents in Systems Manager Incident Manager

> **Note**
>
> All the underlying alarms in your composite alarm must be in the same account and the same Region as your composite alarm. However, if you set up a composite alarm in a CloudWatch cross-account observability monitoring account, the underlying alarms can watch metrics in different source accounts and in the monitoring account itself. For more information, see [CloudWatch cross-account observability]().
> A single composite alarm can monitor 100 underlying alarms, and 150 composite alarms can monitor a single underlying alarm.

Rule expressions

All composite alarms contain rule expressions. Rule expressions tell composite alarms which other alarms to monitor and determine their states from. Rule expressions can refer to metric alarms and composite alarms. When you reference an alarm in a rule expression, you designate a function to the alarm that determines which of the following three states the alarm will be in:

- ALARM

  ALARM ("alarm-name or alarm-ARN") is TRUE if the alarm is in ALARM state.
- OK

  OK ("alarm-name or alarm-ARN") is TRUE if the alarm is in OK state.
- INSUFFICIENT_DATA

  INSUFFICIENT_DATA (“alarm-name or alarm-ARN") is TRUE if the named alarm is in INSUFFICIENT_DATA state.

> **Note**
>
> TRUE always evaluates to TRUE, and FALSE always evaluates to FALSE.

Example expressions

The request parameter `AlarmRule` supports the use of the logical operators `AND`, `OR`, and `NOT`, so you can combine multiple functions into a single expressions. The following example expressions show how you can configure the underlying alarms in your composite alarm:

- `ALARM(CPUUtilizationTooHigh) AND ALARM(DiskReadOpsTooHigh)`

  The expression specifies that the composite alarm goes into `ALARM` only if `CPUUtilizationTooHigh` and `DiskReadOpsTooHigh` are in `ALARM`.
- `ALARM(CPUUtilizationTooHigh) AND NOT ALARM(DeploymentInProgress)`

  The expression specifies that the composite alarm goes into `ALARM` if `CPUUtilizationTooHigh` is in `ALARM` and `DeploymentInProgress` is not in `ALARM`. This is an example of a composite alarm that reduces alarm noise during a deployment window.
- `(ALARM(CPUUtilizationTooHigh) OR ALARM(DiskReadOpsTooHigh)) AND OK(NetworkOutTooHigh)`

The expression specifies that the composite alarm goes into ALARM if (ALARM(CPUUtilizationTooHigh) or (DiskReadOpsTooHigh) is in ALARM and (NetworkOutTooHigh) is in OK. This is an example of a composite alarm that reduces alarm noise by not sending you notifications when either of the underlying alarms aren't in ALARM while a network issue is occurring.

Topics

Create a composite alarm

The steps in this section explain how to use the CloudWatch console to create a composite alarm. You can also use the API or Amazon CLI to create a composite alarm. For more information, see PutCompositeAlarm or put-composite-alarm

To create a composite alarm

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Alarms**, and then choose **All alarms**.
3. From the list of alarms, select the check box next to each of the existing alarms that you want to reference in your rule expression, and then choose **Create composite alarm**.
4. Under **Specify composite alarm conditions**, specify the rule expression for your new composite alarm.

> **Note**
>
> Automatically, the alarms that you selected from the list of alarms are listed in the **Conditions** box. By default, the ALARM function has been designated to each of your alarms, and each of your alarms is joined by the logical operator OR.

You can use the following substeps to modify your rule expression:

a. You can change the required state for each of your alarms from ALARM to OK or INSUFFICIENT_DATA.

b. You can change the logical operator in your rule expression from `OR` to `AND` or `NOT`, and you can add parentheses to group your functions.

c. You can include other alarms in your rule expression or delete alarms from your rule expression.

Example: Rule expression with conditions

```
(ALARM("CPUUtilizationTooHigh") OR
ALARM("DiskReadOpsTooHigh")) AND
OK("NetworkOutTooHigh")
```

In the example rule expression where the composite alarm goes into `ALARM` when ALARM ("CPUUtilizationTooHigh" or ALARM("DiskReadOpsTooHigh") is in `ALARM` at the same time as OK("NetworkOutTooHigh") is in `OK`.

5. When finished, choose **Next**.

6. Under **Configure actions**, you can choose from the following:

For ***Notification***

- **Select an exisiting SNS topic**, **Create a new SNS topic**, or **Use a topic ARN** to define the SNS topic that will receive the notification.
- **Add notification**, so your alarm can send multiple notifications for the same alarm state or different alarm states.
- **Remove** to stop your alarm from sending notifications or taking actions.

(Optional) To have the alarm invoke a Lambda function when it changes state, choose **Add Lambda action**. Then specify the function name or ARN, and optionally choose a specific version of the function.

For ***Systems Manager action***

- **Add Systems Manager action**, so your alarm can perform an SSM action when it goes into ALARM.

To learn more about Systems Manager actions, see [Configuring CloudWatch to create OpsItems from alarms] in the *Amazon Systems Manager User Guide* and [Incident creation] in the *Incident Manager User Guide*. To create an alarm that performs an SSM Incident Manager action, you must have the correct permissions. For more information, see Identity-based policy examples for Amazon Systems Manager Incident Manager in the *Incident Manager User Guide*.

To have the alarm start an investigation, choose **Add investigation action** and then select your investigation group. For more information about Amazon Q Developer operational investigations, see Amazon Q Developer operational investigations (Preview).

7. When finished, choose **Next**.
8. Under **Add name and description**, enter an alarm name and *optional* description for your new composite alarm. The name must contain only UTF-8 characters, and can't contain ASCII control characters. The description can include markdown formatting, which is displayed only in the alarm **Details** tab in the CloudWatch console. The markdown can be useful to add links to runbooks or other internal resources.
9. When finished, choose **Next**.
10. Under **Preview and create**, confirm your information, and then choose **Create composite alarm**.

> **Note**
>
> You can create a cycle of composite alarms, where one composite alarm and another composite alarm depend on each other. If you find yourself in this scenario, your composite alarms stop being evaluated, and you can't delete your composite alarms because they're dependent on each other. The easiest way to break the cycle of dependecy between your composite alarms is to change the function `AlarmRule` in one of your composite alarms to `False`.

Suppressing composite alarm actions

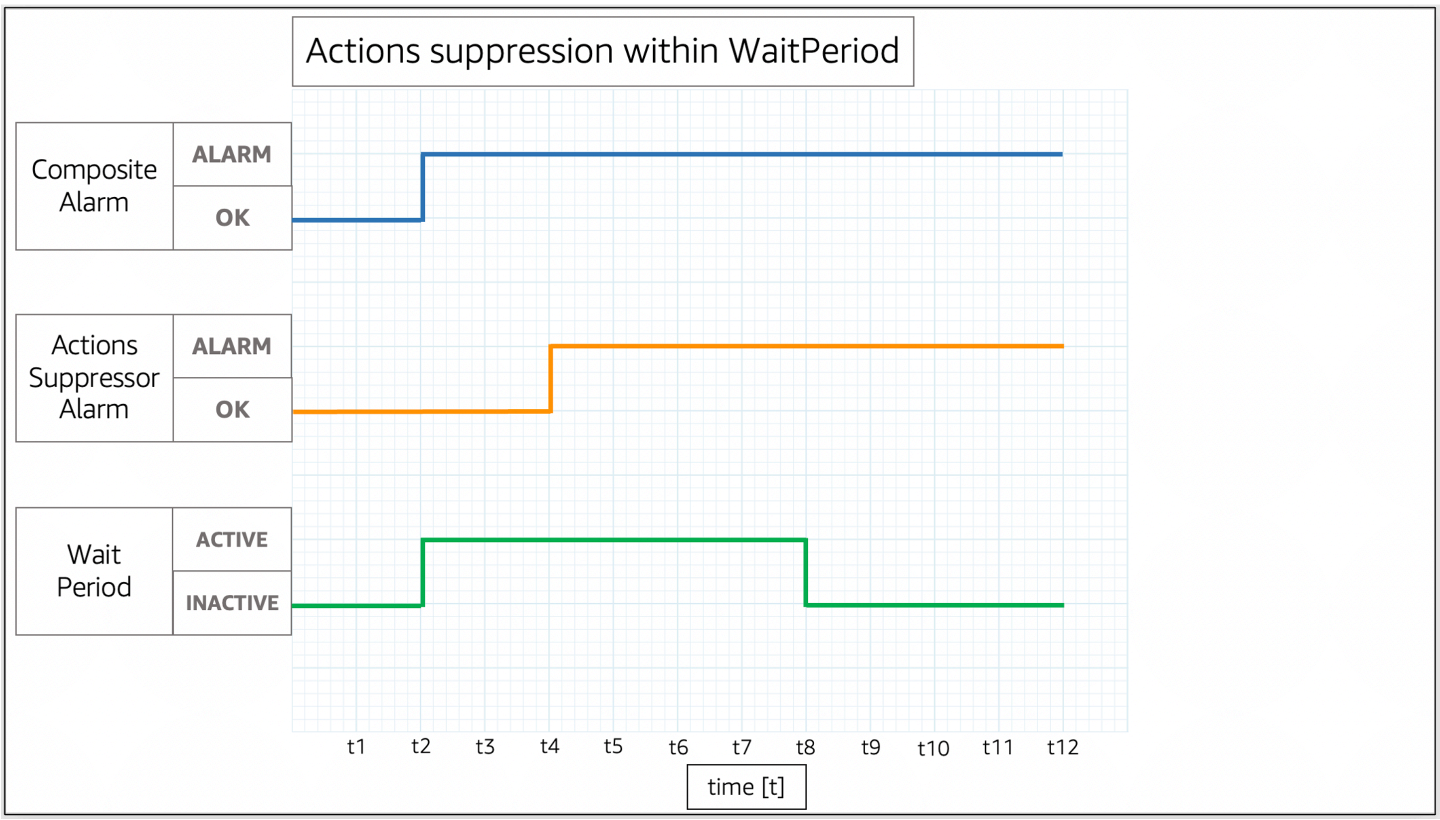
Because composite alarms allow you to get an aggregated view of your health across multiple alarms, there are common situations where it is expected for those alarms to trigger. For example, during a maintenance window of your application or when you investigate an ongoing incident. In such situations, you may want to suppress the actions of your composite alarms, to prevent unwanted notifications or the creation of new incident tickets

With composite alarm action suppression, you define alarms as suppressor alarms. Suppressor alarms prevent composite alarms from taking actions. For example, you can specify a suppressor alarm that represents the status of a supporting resource. If the supporting resource is down, the suppressor alarm prevents the composite alarm from sending notifications. Composite alarm action suppression helps you reduce alarm noise, so you spend less time managing your alarms and more time focusing on your operations.

You specify suppressor alarms when you configure composite alarms. Any alarm can function as a suppressor alarm. When a suppressor alarm changes states from `OK` to `ALARM`, its composite alarm stops taking actions. When a suppressor alarm changes states from `ALARM` to `OK`, its composite alarm resumes taking actions.

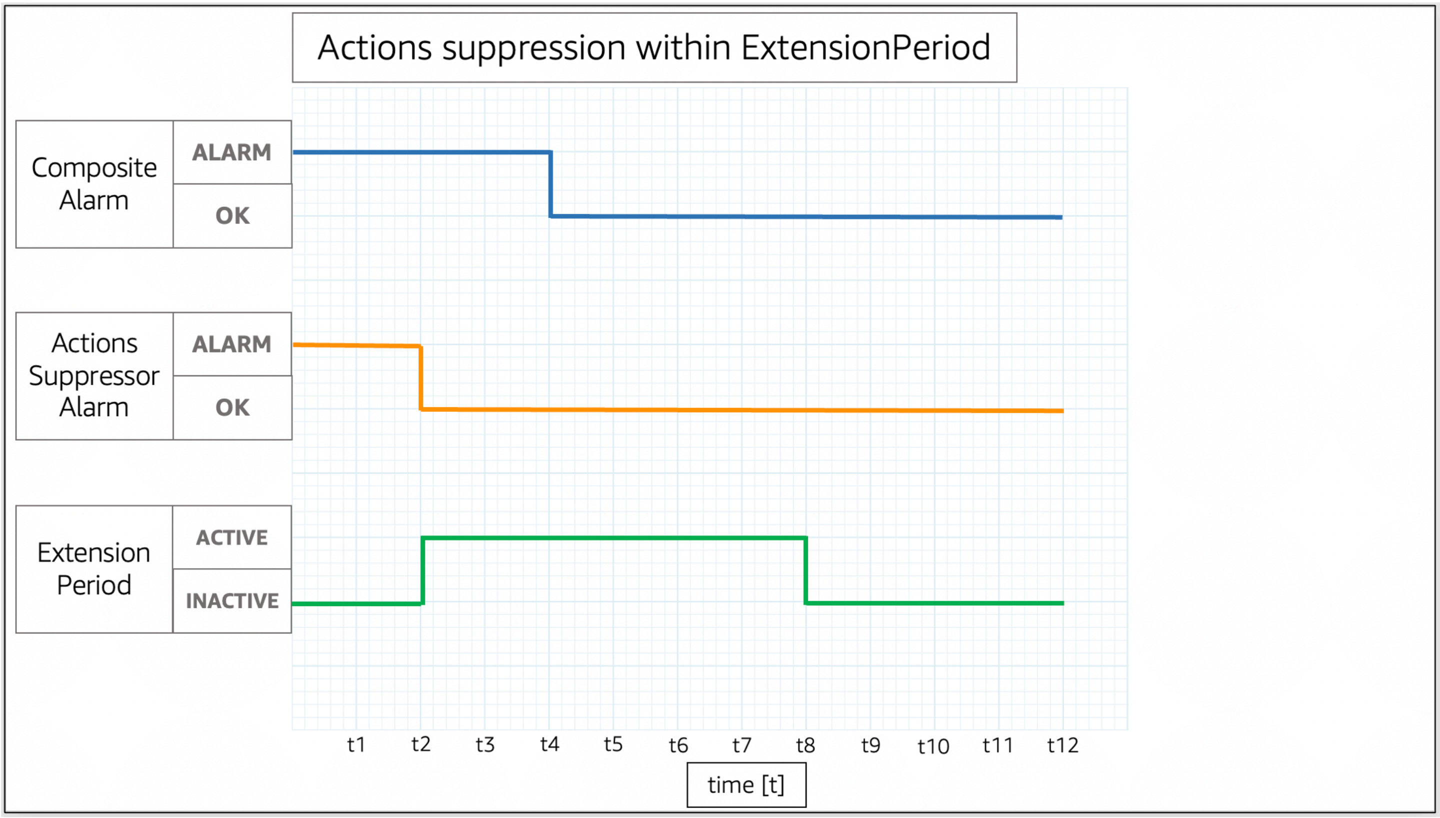
`WaitPeriod` and `ExtensionPeriod`

When you specify a suppressor alarm, you set the parameters `WaitPeriod` and `ExtensionPeriod`. These parameters prevent composite alarms from taking actions unexpectedly while suppressor alarms change states. Use `WaitPeriod` to compensate for any delays that can occur when a suppressor alarm changes from `OK` to `ALARM`. For example, if a suppressor alarm changes from `OK` to `ALARM` within 60 seconds, set `WaitPeriod` to 60 seconds.



In the image, the composite alarm changes from `OK` to `ALARM` at t2. A `WaitPeriod` starts at t2 and ends at t8. This gives the suppressor alarm time to change states from `OK` to `ALARM` at t4 before it suppresses the composite alarm's actions when the `WaitPeriod` expires at t8.

Use `ExtensionPeriod` to compensate for any delays that can occur when a composite alarm changes to `OK` following a suppressor alarm changing to `OK`. For example, if a composite alarm changes to `OK` within 60 seconds of a suppressor alarm changing to `OK`, set `ExtensionPeriod` to 60 seconds.



In the image, the suppressor alarm changes from `ALARM` to `OK` at t2. An `ExtensionPeriod` starts at t2 and ends at t8. This gives the composite alarm time to change from `ALARM` to `OK` before the `ExtensionPeriod` expires at t8.

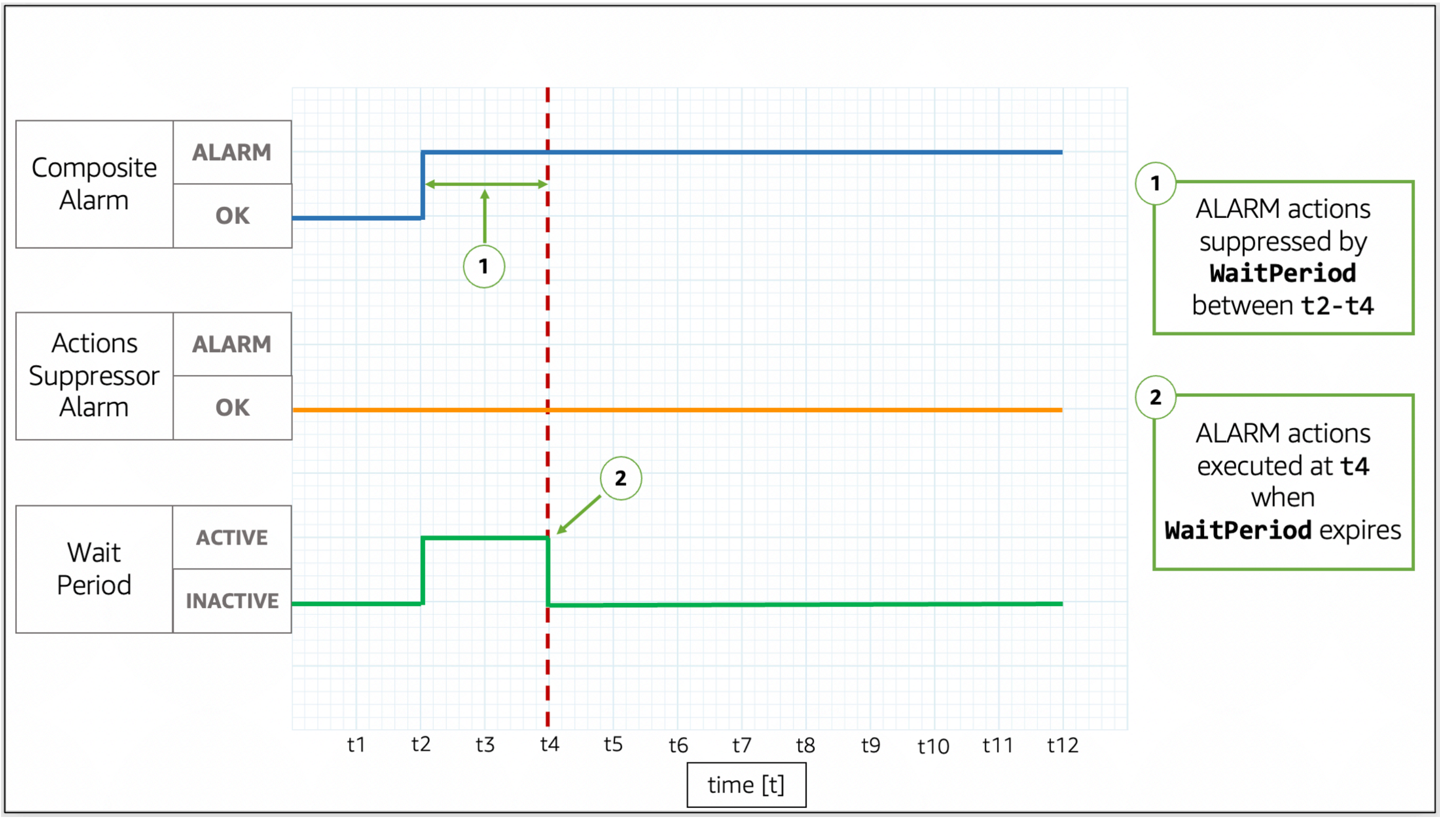
Composite alarms don't take actions when `WaitPeriod` and `ExtensionPeriod` become active. Composite alarms take actions that are based on their currents states when `ExtensionPeriod` and `WaitPeriod` become inactive. We recommend that you set the value for each parameter to 60 seconds, as CloudWatch evaluates metric alarms every minute. You can set the parameters to any integer in seconds.

The following examples describe in more detail how `WaitPeriod` and `ExtensionPeriod` prevent composite alarms from taking actions unexpectedly.

> **Note**
>
> In the following examples, `WaitPeriod` is configured as 2 time units, and `ExtensionPeriod` is configured as 3 time units.

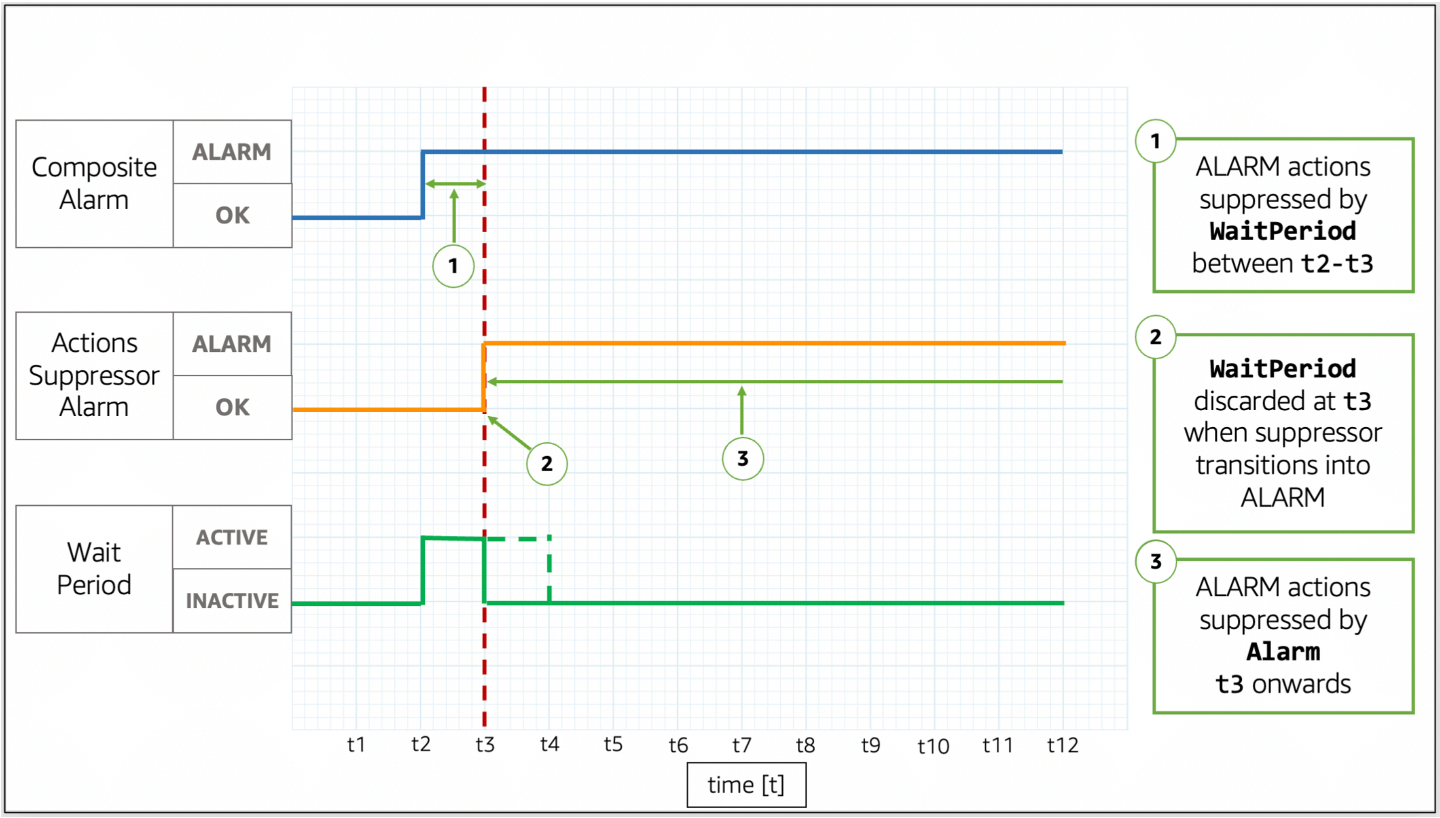
Examples

Example 1: Actions are not suppressed after `WaitPeriod`



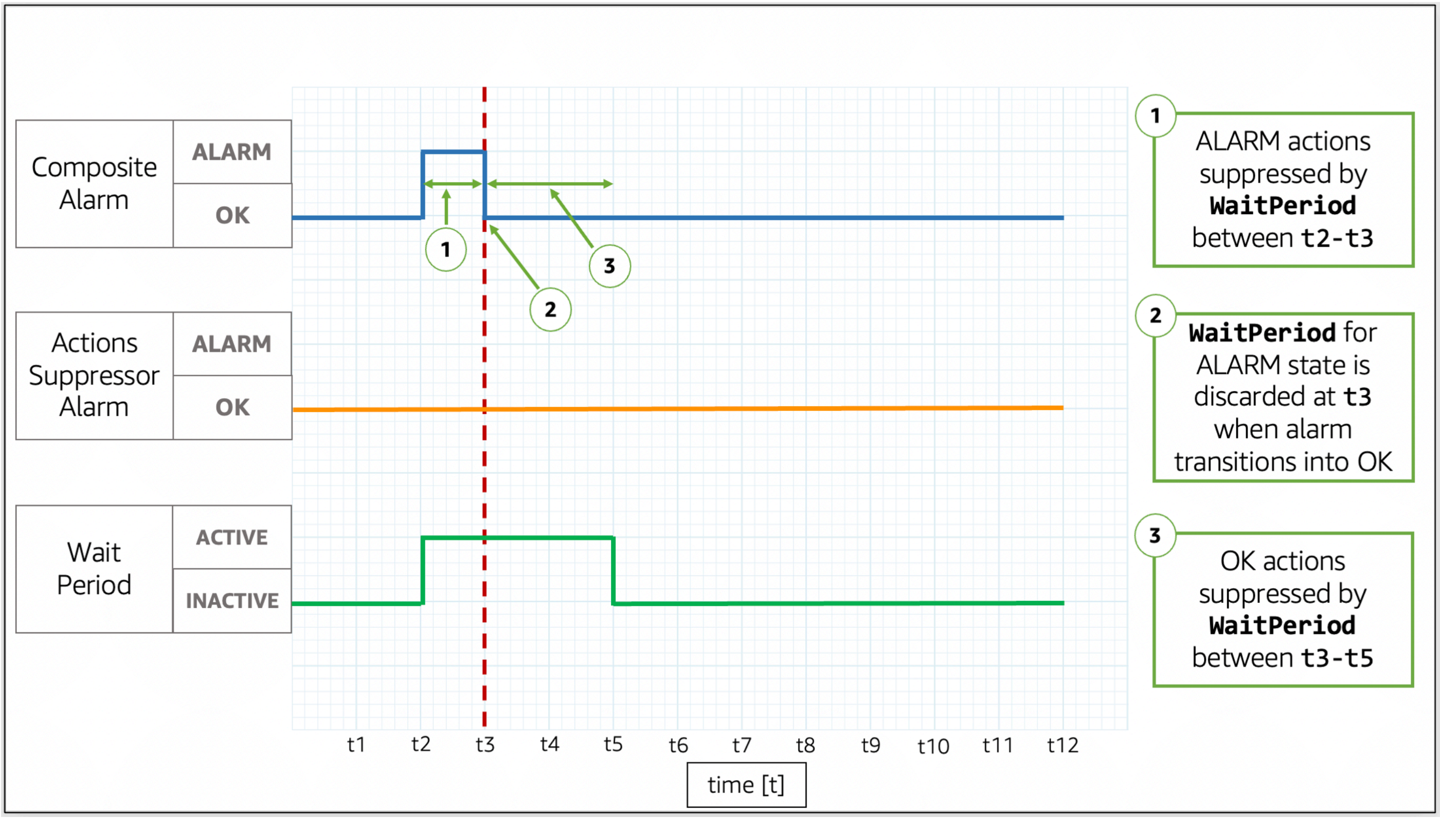
In the image, the composite alarm changes states from `OK` to `ALARM` at t2. A `WaitPeriod` starts at t2 and ends at t4, so it can prevent the composite alarm from taking actions. After the `WaitPeriod` expires at t4, the composite alarm takes its actions because the suppressor alarm is still in `OK`.

Example 2: Actions are suppressed by alarm before `WaitPeriod` expires



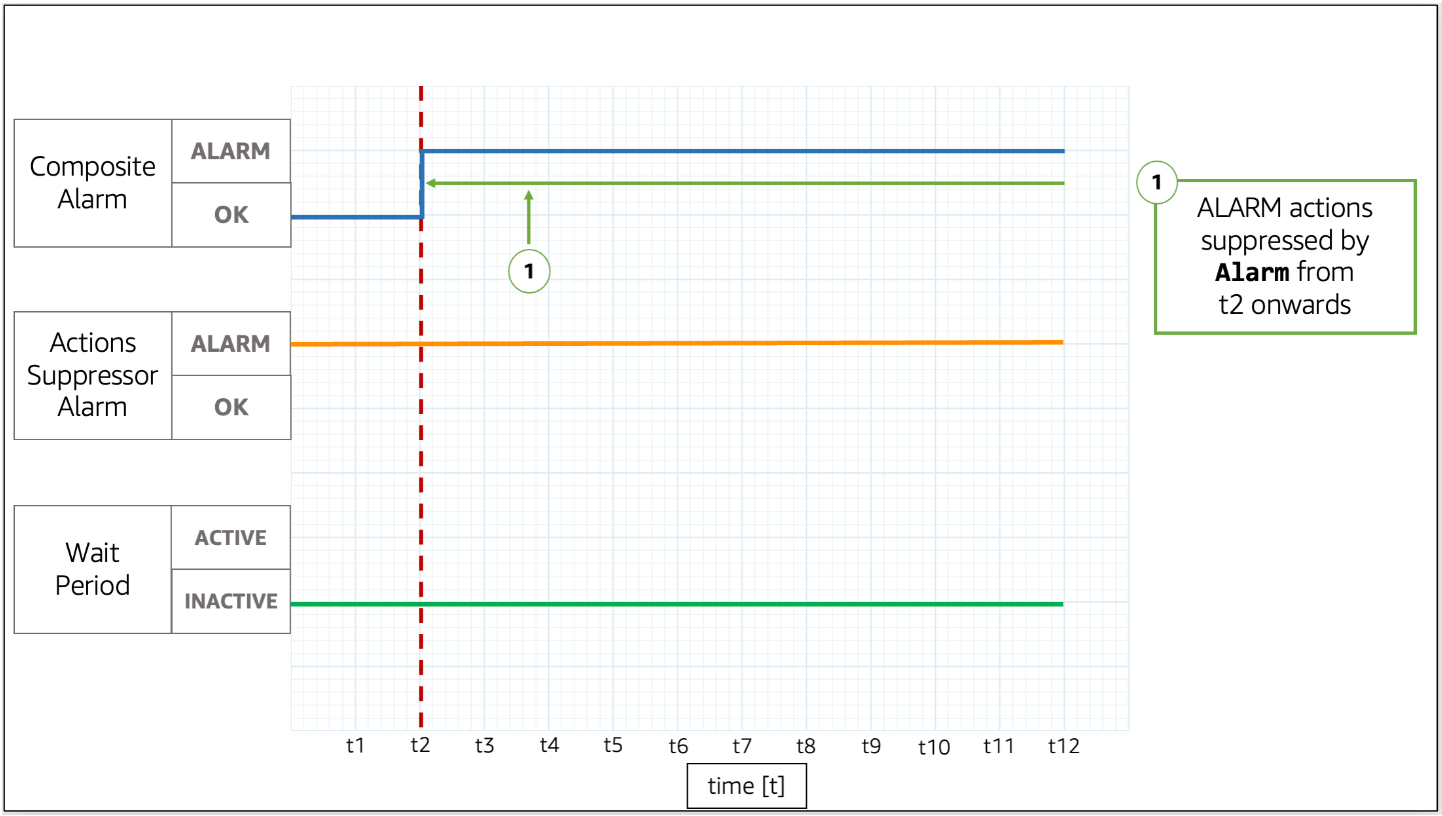
In the image, the composite alarm changes states from `OK` to `ALARM` at t2. A `WaitPeriod` starts at t2 and ends at t4. This gives the suppressor alarm time to change states from `OK` to `ALARM` at t3. Because the suppressor alarm changes states from `OK` to `ALARM` at t3, the `WaitPeriod` that started at t2 is discarded, and the suppressor alarm now stops the composite alarm from taking actions.

Example 3: State transition when actions are suppressed by `WaitPeriod`



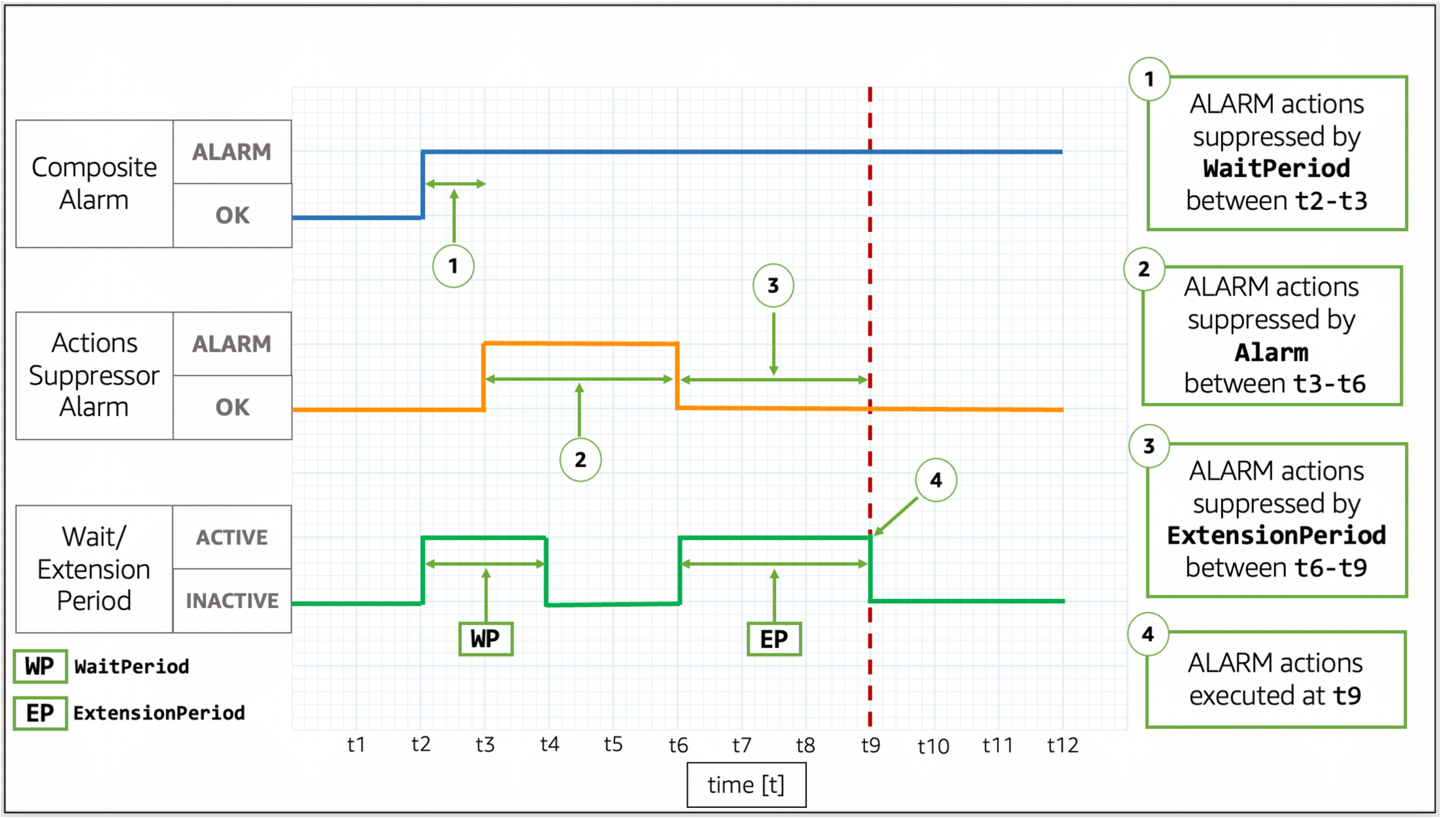
In the image, the composite alarm changes states from `OK` to `ALARM` at t2. A `WaitPeriod` starts at t2 and ends at t4. This gives the suppressor alarm time to change states. The composite alarm changes back to `OK` at t3, so the `WaitPeriod` that started at t2 is discarded. A new `WaitPeriod` starts at t3 and ends at t5. After the new `WaitPeriod` expires at t5, the composite alarm takes its actions.

Example 4: State transition when actions are suppressed by alarm



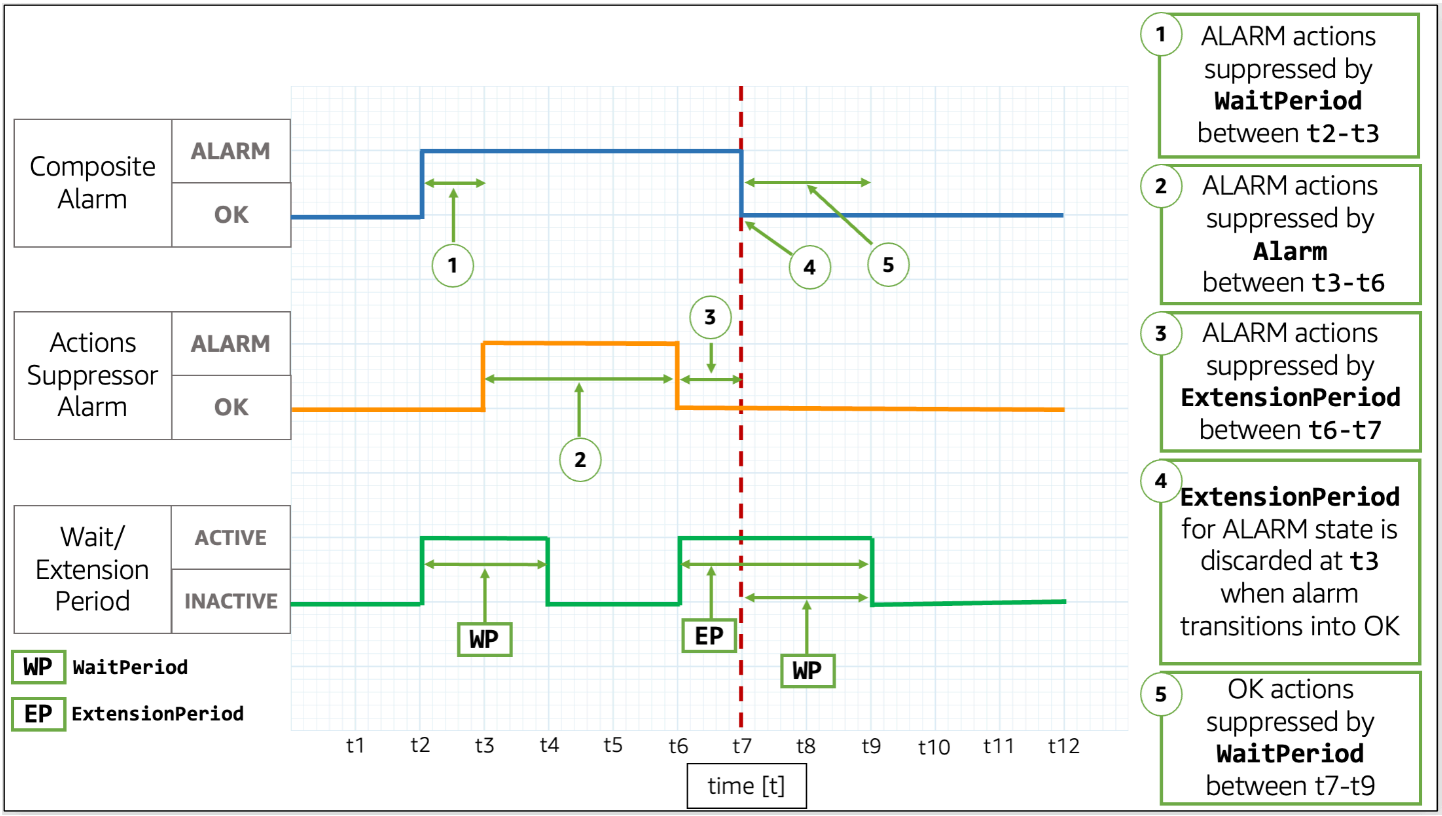
In the image, the composite alarm changes states from `OK` to `ALARM` at t2. The suppressor alarm is already in `ALARM`. The suppressor alarm stops the composite alarm from taking actions.

Example 5: Actions are not suppressed after `ExtensionPeriod`



In the image, the composite alarm changes states from `OK` to `ALARM` at t2. A `WaitPeriod` starts at t2 and ends at t4. This gives the suppressor alarm time to change states from `OK` to `ALARM` at t3 before it suppresses the composite alarm's actions until t6. Because the suppressor alarm changes states from `OK` to `ALARM` at t3, the `WaitPeriod` that started at t2 is discarded. At t6, the suppressor alarm changes to `OK`. An `ExtensionPeriod` starts at t6 and ends at t9. After the `ExtensionPeriod` expires, the composite alarm takes its actions.

Example 6: State transition when actions are suppressed by `ExtensionPeriod`



In the image, the composite alarm changes states from `OK` to `ALARM` at t2. A `WaitPeriod` starts at t2 and ends at t4. This gives the suppressor alarm time to change states from `OK` to `ALARM` at t3 before it suppresses the composite alarm's actions until t6. Because the suppressor alarm changes states from `OK` to `ALARM` at t3, the `WaitPeriod` that started at t2 is discarded. At t6, the suppressor alarm changes back to `OK`. An `ExtensionPeriod` starts at t6 and ends at t9. When the composite alarm changes back to `OK` at t7, the `ExtensionPeriod` is discarded, and a new `WaitPeriod` starts at t7 and ends at t9.

> **Tip**
>
> If you replace the action suppressor alarm, any active `WaitPeriod` or `ExtensionPeriod` is discarded.

Acting on alarm changes

CloudWatch can notify users on two types of alarm changes: when an alarm changes state, and when the configuration of an alarm gets updated.

When an alarm evaluates, it might change from one state to another, such as ALARM, OK or INSUFFICIENT_DATA. These alarm state changes can signal a possible incident, a return to normal, or a metric being unavailable. In such cases, you might want to the alarm to automatically take one of the following actions

- Send a notification to a SNS topic as part of the alarm's actions. An SNS topic can then be configured for application-to-application (A2A) messaging as well as application-to-person (A2P) notifications, including channels such as email notifications and SMS. All the destinations that you define for your SNS topic receive the alarm notification. For more information, see Amazon SNS event destinations.
- You can configure notifications for alarm state change events. Amazon User Notifications offers a native way to configure such notifications and is the recommended approach.
- Reboot, stop, recover, or terminate an Amazon EC2 instance. For more information, see Create alarms to stop, terminate, reboot, or recover an EC2 instance.
- Scale the capacity of an Amazon EC2 Auto Scaling group. For more information, see Step and simple scaling policies for Amazon EC2 Auto Scaling.
- Start an investigation in Amazon Q operational investigations. For more information, see Start an investigation in Amazon Q operational investigations from an alarm
- Invoke a Lambda function.
- Create an incident in Systems Manager Incident Manager. For more information, see Creating incidents automatically with CloudWatch alarms.
- Create an OpsItem in Systems Manager OpsCenter. For more information, see Configure CloudWatch alarms to create OpsItems.

Additionally, CloudWatch sends events to Amazon EventBridge whenever alarms change state, and when alarms are created, deleted, or updated. You can write EventBridge rules to take actions or be notified when EventBridge receives these events.

Topics

Notifying users on alarm changes

This section explains how you can use Amazon User Notifications or Amazon Simple Notification Service to have users be notified of alarm changes.

Setting up Amazon User Notifications

You can use Amazon User Notifications to set up delivery channels to get notified about CloudWatch alarm state change and configuration change events. You receive a notification when an event matches a rule that you specify. You can receive notifications for events through multiple channels, including email, Amazon Chatbot chat notifications, or Amazon Console Mobile Application push notifications. You can also see notifications in the at Console Notifications Center. User Notifications supports aggregation, which can reduce the number of notifications you receive during specific events.

The notification configurations you create with Amazon User Notifications do not count towards the limit on the number of actions you can configure per target alarm state. As Amazon User Notifications matches the events emitted to Amazon EventBridge, it sends notifications for all the alarms in your account and selected Regions, unless you specify an advanced filter to allowlist or denylist specific alarms or patterns.

The following example of an advanced filter matches an alarm state change from OK to ALARM on the alarm named `ServerCpuTooHigh`.

```
{
"detail": {
    "alarmName": ["ServerCpuTooHigh"],
    "previousState": { "value": ["OK"] },
    "state": { "value": ["ALARM"] }
  }
}
```

You can use any of the properties published by an alarm in EventBridge events to create a filter. For more information, see Alarm events and EventBridge.

Setting up Amazon SNS notifications

You can use Amazon Simple Notification Service to send both application-to-application (A2A) messaging and application-to-person (A2P) messaging, including mobile text messaging (SMS) and email messages. For more information, see Amazon SNS event destinations.

For every state that an alarm can take, you can configure the alarm to send a message to an SNS topic. Every Amazon SNS topic you configure for a state on a given alarm counts towards the limit on the number of actions you can configure for that alarm and state. You can send messages to the same Amazon SNS topic from any alarms in your account, and use the same Amazon SNS topic for both application (A2A) and person (A2P) consumers. Because this configuration is done at the alarm level, only the alarms you have configured send messages to the selected Amazon SNS topic.

First, create a topic, then subscribe to it. You can optionally publish a test message to the topic. For an example, see Setting up an Amazon SNS topic using the Amazon Web Services Management Console. Or for more information, see Getting started with Amazon SNS.

Alternatively, if you plan to use the Amazon Web Services Management Console to create your CloudWatch alarm, you can skip this procedure because you can create the topic when you create the alarm.

When you create a CloudWatch alarm, you can add actions for any target state the alarm enters. Add an Amazon SNS notification for the state you want to be notified about, and select the Amazon SNS topic you created in the previous step to send an email notification when the alarm enters the selected state.

> **Note**
>
> When you create an Amazon SNS topic, you choose to make it a *standard topic* or a *FIFO topic*. CloudWatch guarantees the publication of all alarm notifications to both types of topics. However, even if you use a FIFO topic, in rare cases CloudWatch sends the notifications to the topic out of order. If you use a FIFO topic, the alarm sets the message group ID of the alarm notifications to be a hash of the ARN of the alarm.

Topics

- Preventing confused deputy security issues
- Setting up an Amazon SNS topic using the Amazon Web Services Management Console
- Setting up an SNS topic using the Amazon CLI

Preventing confused deputy security issues

The confused deputy problem is a security issue where an entity that doesn't have permission to perform an action can coerce a more-privileged entity to perform the action. In Amazon, cross-service impersonation can result in the confused deputy problem. Cross-service impersonation can occur when one service (the *calling service*) calls another service (the *called service*). The calling service can be manipulated to use its permissions to act on another customer's resources in a way it should not otherwise have permission to access. To prevent this, Amazon provides tools that help you protect your data for all services with service principals that have been given access to resources in your account.

We recommend using the `aws:SourceArn`, `aws:SourceAccount`, `aws:SourceOrgID`, and `aws:SourceOrgPaths` global condition context keys in resource policies to limit the permissions that Amazon SNS gives another service to the resource. Use `aws:SourceArn` to associate only one resource with cross-service access. Use `aws:SourceAccount` to let any resource in that account be associated with the cross-service use. Use `aws:SourceOrgID` to allow any resource from any account within an organization be associated with the cross-service use. Use `aws:SourceOrgPaths` to associate any resource from accounts within an Amazon Organizations path with the cross-service use. For more information about using and understanding paths, see aws:SourceOrgPaths in the IAM User Guide.

The most effective way to protect against the confused deputy problem is to use the `aws:SourceArn` global condition context key with the full ARN of the resource. If you don't know the full ARN of the resource or if you are specifying multiple resources, use the `aws:SourceArn` global context condition key with wildcard characters (*) for the unknown portions of the ARN. For example, `arn:aws-cn:servicename:*:123456789012:*`.

If the `aws:SourceArn` value does not contain the account ID, such as an Amazon S3 bucket ARN, you must use both `aws:SourceAccount` and `aws:SourceArn` to limit permissions.

To protect against the confused deputy problem at scale, use the `aws:SourceOrgID` or `aws:SourceOrgPaths` global condition context key with the organization ID or organization path of the resource in your resource-based policies. Policies that include the `aws:SourceOrgID` or `aws:SourceOrgPaths` key will automatically include the correct accounts and you don't have to manually update the policies when you add, remove, or move accounts in your organization.

The value of `aws:SourceArn` must be the ARN of the alarm that is sending notifications.

The following example shows how you can use the `aws:SourceArn` and `aws:SourceAccount` global condition context keys in CloudWatch to prevent the confused deputy problem.

```
{
    "Statement": [{
        "Effect": "Allow",
```

```
        "Principal": {
            "Service": "cloudwatch.amazonaws.com"
        },
        "Action": "SNS:Publish",
        "Resource": "arn:aws:sns:us-east-2:444455556666:MyTopic",
        "Condition": {
            "ArnLike": {
                "aws:SourceArn": "arn:aws:cloudwatch:us-east-2:111122223333:alarm:*"
            },
            "StringEquals": {
                "aws:SourceAccount": "111122223333"
            }
        }
    }]
}
```

If an alarm ARN includes any non-ASCII characters, use only the `aws:SourceAccount` global condition key to limit the permissions.

Setting up an Amazon SNS topic using the Amazon Web Services Management Console

First, create a topic, then subscribe to it. You can optionally publish a test message to the topic.

To create an SNS topic

1. Open the Amazon SNS console at https://console.amazonaws.cn/sns/v3/home.
2. On the Amazon SNS dashboard, under **Common actions**, choose **Create Topic**.
3. In the **Create new topic** dialog box, for **Topic name**, enter a name for the topic (for example, **`my-topic`**).
4. Choose **Create topic**.
5. Copy the **Topic ARN** for the next task (for example, arn:aws:sns:us-east-1:111122223333:my-topic).

To subscribe to an SNS topic

1. Open the Amazon SNS console at https://console.amazonaws.cn/sns/v3/home.
2. In the navigation pane, choose **Subscriptions**, **Create subscription**.
3. In the **Create subscription** dialog box, for **Topic ARN**, paste the topic ARN that you created in the previous task.

4. For **Protocol**, choose **Email**.
5. For **Endpoint**, enter an email address that you can use to receive the notification, and then choose **Create subscription**.
6. From your email application, open the message from Amazon Notifications and confirm your subscription.

   Your web browser displays a confirmation response from Amazon SNS.

To publish a test message to an SNS topic

1. Open the Amazon SNS console at https://console.amazonaws.cn/sns/v3/home.
2. In the navigation pane, choose **Topics**.
3. On the **Topics** page, select a topic and choose **Publish to topic**.
4. In the **Publish a message** page, for **Subject**, enter a subject line for your message, and for **Message**, enter a brief message.
5. Choose **Publish Message**.
6. Check your email to confirm that you received the message.

Setting up an SNS topic using the Amazon CLI

First, you create an SNS topic, and then you publish a message directly to the topic to test that you have properly configured it.

To set up an SNS topic

1. Create the topic using the create-topic command as follows.

   ```
   aws sns create-topic --name my-topic
   ```

   Amazon SNS returns a topic ARN with the following format:

   ```
   {
       "TopicArn": "arn:aws:sns:us-east-1:111122223333:my-topic"
   }
   ```

2. Subscribe your email address to the topic using the subscribe command. If the subscription request succeeds, you receive a confirmation email message.

```
aws sns subscribe --topic-arn arn:aws:sns:us-east-1:111122223333:my-topic --
protocol email --notification-endpoint my-email-address
```

Amazon SNS returns the following:

```
{
    "SubscriptionArn": "pending confirmation"
}
```

3. From your email application, open the message from Amazon Notifications and confirm your subscription.

   Your web browser displays a confirmation response from Amazon Simple Notification Service.

4. Check the subscription using the list-subscriptions-by-topic command.

```
aws sns list-subscriptions-by-topic --topic-arn arn:aws:sns:us-
east-1:111122223333:my-topic
```

   Amazon SNS returns the following:

```
{
  "Subscriptions": [
    {
        "Owner": "111122223333",
        "Endpoint": "me@mycompany.com",
        "Protocol": "email",
        "TopicArn": "arn:aws:sns:us-east-1:111122223333:my-topic",
        "SubscriptionArn": "arn:aws:sns:us-east-1:111122223333:my-topic:64886986-
bf10-48fb-a2f1-dab033aa67a3"
    }
  ]
}
```

5. (Optional) Publish a test message to the topic using the publish command.

```
aws sns publish --message "Verification" --topic arn:aws:sns:us-
east-1:111122223333:my-topic
```

   Amazon SNS returns the following.

```
{
    "MessageId": "42f189a0-3094-5cf6-8fd7-c2dde61a4d7d"
}
```

6. Check your email to confirm that you received the message.

Schema of Amazon SNS notifications when alarms change state

This section lists the schemas of the notifications sent to Amazon SNS topics when alarms change their state.

Schema when a metric alarm changes state

```
{
  "AlarmName": "string",
  "AlarmDescription": "string",
  "AWSAccountId": "string",
  "AlarmConfigurationUpdatedTimestamp": "string",
  "NewStateValue": "string",
  "NewStateReason": "string",
  "StateChangeTime": "string",
  "Region": "string",
  "AlarmArn": "string",
  "OldStateValue": "string",
  "OKActions": ["string"],
  "AlarmActions": ["string"],
  "InsufficientDataActions": ["string"],
  "Trigger": {
    "MetricName": "string",
    "Namespace": "string",
    "StatisticType": "string",
    "Statistic": "string",
    "Unit": "string or null",
    "Dimensions": [
      {
        "value": "string",
        "name": "string"
      }
    ],
    "Period": "integer",
    "EvaluationPeriods": "integer",
    "DatapointsToAlarm": "integer",
```

```
    "ComparisonOperator": "string",
    "Threshold": "number",
    "TreatMissingData": "string",
    "EvaluateLowSampleCountPercentile": "string or null"
  }
}
```

Schema when a composite alarm changes state

```
{
  "AlarmName": "string",
  "AlarmDescription": "string",
  "AWSAccountId": "string",
  "NewStateValue": "string",
  "NewStateReason": "string",
  "StateChangeTime": "string",
  "Region": "string",
  "AlarmArn": "string",
  "OKActions": [String],
  "AlarmActions": [String],
  "InsufficientDataActions": [String],
  "OldStateValue": "string",
  "AlarmRule": "string",
  "TriggeringChildren": [String]
}
```

Alarm events and EventBridge

CloudWatch sends events to Amazon EventBridge whenever a CloudWatch alarm is created, updated, deleted, or changes alarm state. You can use EventBridge and these events to write rules that take actions, such as notifying you, when an alarm changes state. For more information, see [What is Amazon EventBridge?](#)

CloudWatch guarantees the delivery of alarm state change events to EventBridge.

Sample events from CloudWatch

This section includes example events from CloudWatch.

State change for a single-metric alarm

```
{
    "version": "0",
```

```
    "id": "c4c1c1c9-6542-e61b-6ef0-8c4d36933a92",
    "detail-type": "CloudWatch Alarm State Change",
    "source": "aws.cloudwatch",
    "account": "123456789012",
    "time": "2019-10-02T17:04:40Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:cloudwatch:us-east-1:123456789012:alarm:ServerCpuTooHigh"
    ],
    "detail": {
        "alarmName": "ServerCpuTooHigh",
        "configuration": {
            "description": "Goes into alarm when server CPU utilization is too high!",
            "metrics": [
                {
                    "id": "30b6c6b2-a864-43a2-4877-c09a1afc3b87",
                    "metricStat": {
                        "metric": {
                            "dimensions": {
                                "InstanceId": "i-12345678901234567"
                            },
                            "name": "CPUUtilization",
                            "namespace": "AWS/EC2"
                        },
                        "period": 300,
                        "stat": "Average"
                    },
                    "returnData": true
                }
            ]
        },
        "previousState": {
            "reason": "Threshold Crossed: 1 out of the last 1 datapoints
 [0.0666851903306472 (01/10/19 13:46:00)] was not greater than the threshold (50.0)
 (minimum 1 datapoint for ALARM -> OK transition).",
            "reasonData": "{\"version\":\"1.0\",\"queryDate\":
\"2019-10-01T13:56:40.985+0000\",\"startDate\":\"2019-10-01T13:46:00.000+0000\",
\"statistic\":\"Average\",\"period\":300,\"recentDatapoints\":[0.0666851903306472],
\"threshold\":50.0}",
            "timestamp": "2019-10-01T13:56:40.987+0000",
            "value": "OK"
        },
        "state": {
```

```
            "reason": "Threshold Crossed: 1 out of the last 1 datapoints
 [99.50160229693434 (02/10/19 16:59:00)] was greater than the threshold (50.0) (minimum
 1 datapoint for OK -> ALARM transition).",
            "reasonData": "{\"version\":\"1.0\",\"queryDate\":
\"2019-10-02T17:04:40.985+0000\",\"startDate\":\"2019-10-02T16:59:00.000+0000\",
\"statistic\":\"Average\",\"period\":300,\"recentDatapoints\":[99.50160229693434],
\"threshold\":50.0}",
            "timestamp": "2019-10-02T17:04:40.989+0000",
            "value": "ALARM"
        }
    }
}
```


State change for a metric math alarm


```
{
    "version": "0",
    "id": "2dde0eb1-528b-d2d5-9ca6-6d590caf2329",
    "detail-type": "CloudWatch Alarm State Change",
    "source": "aws.cloudwatch",
    "account": "123456789012",
    "time": "2019-10-02T17:20:48Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:cloudwatch:us-east-1:123456789012:alarm:TotalNetworkTrafficTooHigh"
    ],
    "detail": {
        "alarmName": "TotalNetworkTrafficTooHigh",
        "configuration": {
            "description": "Goes into alarm if total network traffic exceeds 10Kb",
            "metrics": [
                {
                    "expression": "SUM(METRICS())",
                    "id": "e1",
                    "label": "Total Network Traffic",
                    "returnData": true
                },
                {
                    "id": "m1",
                    "metricStat": {
                        "metric": {
                            "dimensions": {
                                "InstanceId": "i-12345678901234567"
```

```
                            },
                            "name": "NetworkIn",
                            "namespace": "AWS/EC2"
                        },
                        "period": 300,
                        "stat": "Maximum"
                    },
                    "returnData": false
                },
                {
                    "id": "m2",
                    "metricStat": {
                        "metric": {
                            "dimensions": {
                                "InstanceId": "i-12345678901234567"
                            },
                            "name": "NetworkOut",
                            "namespace": "AWS/EC2"
                        },
                        "period": 300,
                        "stat": "Maximum"
                    },
                    "returnData": false
                }
            ]
        },
        "previousState": {
            "reason": "Unchecked: Initial alarm creation",
            "timestamp": "2019-10-02T17:20:03.642+0000",
            "value": "INSUFFICIENT_DATA"
        },
        "state": {
            "reason": "Threshold Crossed: 1 out of the last 1 datapoints [45628.0
 (02/10/19 17:10:00)] was greater than the threshold (10000.0) (minimum 1 datapoint for
 OK -> ALARM transition).",
            "reasonData": "{\"version\":\"1.0\",\"queryDate\":
\"2019-10-02T17:20:48.551+0000\",\"startDate\":\"2019-10-02T17:10:00.000+0000\",
\"period\":300,\"recentDatapoints\":[45628.0],\"threshold\":10000.0}",
            "timestamp": "2019-10-02T17:20:48.554+0000",
            "value": "ALARM"
        }
    }
}
```

State change for an anomaly detection alarm


```
{
    "version": "0",
    "id": "daafc9f1-bddd-c6c9-83af-74971fcfc4ef",
    "detail-type": "CloudWatch Alarm State Change",
    "source": "aws.cloudwatch",
    "account": "123456789012",
    "time": "2019-10-03T16:00:04Z",
    "region": "us-east-1",
    "resources": ["arn:aws:cloudwatch:us-east-1:123456789012:alarm:EC2 CPU Utilization 
 Anomaly"],
    "detail": {
        "alarmName": "EC2 CPU Utilization Anomaly",
        "state": {
            "value": "ALARM",
            "reason": "Thresholds Crossed: 1 out of the last 1 datapoints [0.0 
 (03/10/19 15:58:00)] was less than the lower thresholds [0.020599444741798756] or 
 greater than the upper thresholds [0.3006915352732461] (minimum 1 datapoint for OK -> 
 ALARM transition).",
            "reasonData": "{\"version\":\"1.0\",\"queryDate\":
\"2019-10-03T16:00:04.650+0000\",\"startDate\":\"2019-10-03T15:58:00.000+0000\",
\"period\":60,\"recentDatapoints\":[0.0],\"recentLowerThresholds\":
[0.020599444741798756],\"recentUpperThresholds\":[0.3006915352732461]}",
            "timestamp": "2019-10-03T16:00:04.653+0000"
        },
        "previousState": {
            "value": "OK",
            "reason": "Thresholds Crossed: 1 out of the last 1 datapoints 
 [0.166666666664241 (03/10/19 15:57:00)] was not less than the lower thresholds 
 [0.0206719426210418] or not greater than the upper thresholds [0.30076870222143803] 
 (minimum 1 datapoint for ALARM -> OK transition).",
            "reasonData": "{\"version\":\"1.0\",\"queryDate\":
\"2019-10-03T15:59:04.670+0000\",\"startDate\":\"2019-10-03T15:57:00.000+0000\",
\"period\":60,\"recentDatapoints\":[0.166666666664241],\"recentLowerThresholds\":
[0.0206719426210418],\"recentUpperThresholds\":[0.30076870222143803]}",
            "timestamp": "2019-10-03T15:59:04.672+0000"
        },
        "configuration": {
            "description": "Goes into alarm if CPU Utilization is out of band",
            "metrics": [{
                "id": "m1",
                "metricStat": {
                    "metric": {
```

```
                        "namespace": "AWS/EC2",
                        "name": "CPUUtilization",
                        "dimensions": {
                            "InstanceId": "i-12345678901234567"
                        }
                    },
                    "period": 60,
                    "stat": "Average"
                },
                "returnData": true
            }, {
                "id": "ad1",
                "expression": "ANOMALY_DETECTION_BAND(m1, 0.8)",
                "label": "CPUUtilization (expected)",
                "returnData": true
            }]
        }
    }
}
```


State change for a composite alarm with a suppressor alarm


```
{
    "version": "0",
    "id": "d3dfc86d-384d-24c8-0345-9f7986db0b80",
    "detail-type": "CloudWatch Alarm State Change",
    "source": "aws.cloudwatch",
    "account": "123456789012",
    "time": "2022-07-22T15:57:45Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:cloudwatch:us-east-1:123456789012:alarm:ServiceAggregatedAlarm"
    ],
    "detail": {
        "alarmName": "ServiceAggregatedAlarm",
        "state": {
            "actionsSuppressedBy": "WaitPeriod",
            "actionsSuppressedReason": "Actions suppressed by WaitPeriod",
            "value": "ALARM",
            "reason": "arn:aws:cloudwatch:us-
east-1:123456789012:alarm:SuppressionDemo.EventBridge.FirstChild transitioned to ALARM
 at Friday 22 July, 2022 15:57:45 UTC",
```

```
            "reasonData": "{\"triggeringAlarms\":[{\"arn\":\"arn:aws:cloudwatch:us-
east-1:123456789012:alarm:ServerCpuTooHigh\",\"state\":{\"value\":\"ALARM\",\"timestamp
\":\"2022-07-22T15:57:45.394+0000\"}}]}",
            "timestamp": "2022-07-22T15:57:45.394+0000"
        },
        "previousState": {
            "value": "OK",
            "reason": "arn:aws:cloudwatch:us-
east-1:123456789012:alarm:SuppressionDemo.EventBridge.Main was created and its alarm
 rule evaluates to OK",
            "reasonData": "{\"triggeringAlarms\":[{\"arn\":\"arn:aws:cloudwatch:us-
east-1:123456789012:alarm:TotalNetworkTrafficTooHigh\",\"state\":{\"value\":\"OK\",
\"timestamp\":\"2022-07-14T16:28:57.770+0000\"}},{\"arn\":\"arn:aws:cloudwatch:us-
east-1:123456789012:alarm:ServerCpuTooHigh\",\"state\":{\"value\":\"OK\",\"timestamp\":
\"2022-07-14T16:28:54.191+0000\"}}]}",
            "timestamp": "2022-07-22T15:56:14.552+0000"
        },
        "configuration": {
            "alarmRule": "ALARM(ServerCpuTooHigh) OR
 ALARM(TotalNetworkTrafficTooHigh)",
            "actionsSuppressor": "ServiceMaintenanceAlarm",
            "actionsSuppressorWaitPeriod": 120,
            "actionsSuppressorExtensionPeriod": 180
        }
    }
}
```


Creation of a composite alarm


```
{

    "version": "0",
    "id": "91535fdd-1e9c-849d-624b-9a9f2b1d09d0",
    "detail-type": "CloudWatch Alarm Configuration Change",
    "source": "aws.cloudwatch",
    "account": "123456789012",
    "time": "2022-03-03T17:06:22Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:cloudwatch:us-east-1:123456789012:alarm:ServiceAggregatedAlarm"
    ],
    "detail": {
        "alarmName": "ServiceAggregatedAlarm",
```

```
        "operation": "create",
        "state": {
            "value": "INSUFFICIENT_DATA",
            "timestamp": "2022-03-03T17:06:22.289+0000"
        },
        "configuration": {
            "alarmRule": "ALARM(ServerCpuTooHigh) OR
 ALARM(TotalNetworkTrafficTooHigh)",
            "alarmName": "ServiceAggregatedAlarm",
            "description": "Aggregated monitor for instance",
            "actionsEnabled": true,
            "timestamp": "2022-03-03T17:06:22.289+0000",
            "okActions": [],
            "alarmActions": [],
            "insufficientDataActions": []
        }
    }
}
```


Creation of a composite alarm with a suppressor alarm


```
{
    "version": "0",
    "id": "454773e1-09f7-945b-aa2c-590af1c3f8e0",
    "detail-type": "CloudWatch Alarm Configuration Change",
    "source": "aws.cloudwatch",
    "account": "123456789012",
    "time": "2022-07-14T13:59:46Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:cloudwatch:us-east-1:123456789012:alarm:ServiceAggregatedAlarm"
    ],
    "detail": {
        "alarmName": "ServiceAggregatedAlarm",
        "operation": "create",
        "state": {
            "value": "INSUFFICIENT_DATA",
            "timestamp": "2022-07-14T13:59:46.425+0000"
        },
        "configuration": {
            "alarmRule": "ALARM(ServerCpuTooHigh) OR
 ALARM(TotalNetworkTrafficTooHigh)",
            "actionsSuppressor": "ServiceMaintenanceAlarm",
```

```
            "actionsSuppressorWaitPeriod": 120,
            "actionsSuppressorExtensionPeriod": 180,
            "alarmName": "ServiceAggregatedAlarm",
            "actionsEnabled": true,
            "timestamp": "2022-07-14T13:59:46.425+0000",
            "okActions": [],
            "alarmActions": [],
            "insufficientDataActions": []
        }
    }
}
```


Update of a metric alarm


```
{

    "version": "0",
    "id": "bc7d3391-47f8-ae47-f457-1b4d06118d50",
    "detail-type": "CloudWatch Alarm Configuration Change",
    "source": "aws.cloudwatch",
    "account": "123456789012",
    "time": "2022-03-03T17:06:34Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:cloudwatch:us-east-1:123456789012:alarm:ServerCpuTooHigh"
    ],
    "detail": {
        "alarmName": "ServerCpuTooHigh",
        "operation": "update",
        "state": {
            "value": "INSUFFICIENT_DATA",
            "timestamp": "2022-03-03T17:06:13.757+0000"
        },
        "configuration": {
            "evaluationPeriods": 1,
            "threshold": 80,
            "comparisonOperator": "GreaterThanThreshold",
            "treatMissingData": "ignore",
            "metrics": [
                {
                    "id": "86bfa85f-b14c-ebf7-8916-7da014ce23c0",
                    "metricStat": {
                        "metric": {
```

```
                        "namespace": "AWS/EC2",
                        "name": "CPUUtilization",
                        "dimensions": {
                            "InstanceId": "i-12345678901234567"
                        }
                    },
                    "period": 300,
                    "stat": "Average"
                },
                "returnData": true
            }
        ],
        "alarmName": "ServerCpuTooHigh",
        "description": "Goes into alarm when server CPU utilization is too high!",
        "actionsEnabled": true,
        "timestamp": "2022-03-03T17:06:34.267+0000",
        "okActions": [],
        "alarmActions": [],
        "insufficientDataActions": []
    },
    "previousConfiguration": {
        "evaluationPeriods": 1,
        "threshold": 70,
        "comparisonOperator": "GreaterThanThreshold",
        "treatMissingData": "ignore",
        "metrics": [
            {
                "id": "d6bfa85f-893e-b052-a58b-4f9295c9111a",
                "metricStat": {
                    "metric": {
                        "namespace": "AWS/EC2",
                        "name": "CPUUtilization",
                        "dimensions": {
                            "InstanceId": "i-12345678901234567"
                        }
                    },
                    "period": 300,
                    "stat": "Average"
                },
                "returnData": true
            }
        ],
        "alarmName": "ServerCpuTooHigh",
        "description": "Goes into alarm when server CPU utilization is too high!",
```

```
            "actionsEnabled": true,
            "timestamp": "2022-03-03T17:06:13.757+0000",
            "okActions": [],
            "alarmActions": [],
            "insufficientDataActions": []
        }
    }
}
```

Update of a composite alarm with a suppressor alarm


```
{
    "version": "0",
    "id": "4c6f4177-6bd5-c0ca-9f05-b4151c54568b",
    "detail-type": "CloudWatch Alarm Configuration Change",
    "source": "aws.cloudwatch",
    "account": "123456789012",
    "time": "2022-07-14T13:59:56Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:cloudwatch:us-east-1:123456789012:alarm:ServiceAggregatedAlarm"
    ],
    "detail": {
        "alarmName": "ServiceAggregatedAlarm",
        "operation": "update",
        "state": {
            "actionsSuppressedBy": "WaitPeriod",
            "value": "ALARM",
            "timestamp": "2022-07-14T13:59:46.425+0000"
        },
        "configuration": {
            "alarmRule": "ALARM(ServerCpuTooHigh) OR
 ALARM(TotalNetworkTrafficTooHigh)",
            "actionsSuppressor": "ServiceMaintenanceAlarm",
            "actionsSuppressorWaitPeriod": 120,
            "actionsSuppressorExtensionPeriod": 360,
            "alarmName": "ServiceAggregatedAlarm",
            "actionsEnabled": true,
            "timestamp": "2022-07-14T13:59:56.290+0000",
            "okActions": [],
            "alarmActions": [],
            "insufficientDataActions": []
        },
```

```
        "previousConfiguration": {
            "alarmRule": "ALARM(ServerCpuTooHigh) OR
 ALARM(TotalNetworkTrafficTooHigh)",
            "actionsSuppressor": "ServiceMaintenanceAlarm",
            "actionsSuppressorWaitPeriod": 120,
            "actionsSuppressorExtensionPeriod": 180,
            "alarmName": "ServiceAggregatedAlarm",
            "actionsEnabled": true,
            "timestamp": "2022-07-14T13:59:46.425+0000",
            "okActions": [],
            "alarmActions": [],
            "insufficientDataActions": []
        }
    }
}
```


Deletion of a metric math alarm


```
{


    "version": "0",
    "id": "f171d220-9e1c-c252-5042-2677347a83ed",
    "detail-type": "CloudWatch Alarm Configuration Change",
    "source": "aws.cloudwatch",
    "account": "123456789012",
    "time": "2022-03-03T17:07:13Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:cloudwatch:us-east-1:123456789012:alarm:TotalNetworkTrafficTooHigh"
    ],
    "detail": {
        "alarmName": "TotalNetworkTrafficTooHigh",
        "operation": "delete",
        "state": {
            "value": "INSUFFICIENT_DATA",
            "timestamp": "2022-03-03T17:06:17.672+0000"
        },
        "configuration": {
            "evaluationPeriods": 1,
            "threshold": 10000,
            "comparisonOperator": "GreaterThanThreshold",
            "treatMissingData": "ignore",
            "metrics": [{
```

```
                    "id": "m1",
                    "metricStat": {
                        "metric": {
                            "namespace": "AWS/EC2",
                            "name": "NetworkIn",
                            "dimensions": {
                                "InstanceId": "i-12345678901234567"
                            }
                        },
                        "period": 300,
                        "stat": "Maximum"
                    },
                    "returnData": false
                },
                {
                    "id": "m2",
                    "metricStat": {
                        "metric": {
                            "namespace": "AWS/EC2",
                            "name": "NetworkOut",
                            "dimensions": {
                                "InstanceId": "i-12345678901234567"
                            }
                        },
                        "period": 300,
                        "stat": "Maximum"
                    },
                    "returnData": false
                },
                {
                    "id": "e1",
                    "expression": "SUM(METRICS())",
                    "label": "Total Network Traffic",
                    "returnData": true
                }
            ],
            "alarmName": "TotalNetworkTrafficTooHigh",
            "description": "Goes into alarm if total network traffic exceeds 10Kb",
            "actionsEnabled": true,
            "timestamp": "2022-03-03T17:06:17.672+0000",
            "okActions": [],
            "alarmActions": [],
            "insufficientDataActions": []
```

```
        }
    }
}
```

Deletion of a composite alarm with a suppressor alarm


```
{
    "version": "0",
    "id": "e34592a1-46c0-b316-f614-1b17a87be9dc",
    "detail-type": "CloudWatch Alarm Configuration Change",
    "source": "aws.cloudwatch",
    "account": "123456789012",
    "time": "2022-07-14T14:00:01Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:cloudwatch:us-east-1:123456789012:alarm:ServiceAggregatedAlarm"
    ],
    "detail": {
        "alarmName": "ServiceAggregatedAlarm",
        "operation": "delete",
        "state": {
            "actionsSuppressedBy": "WaitPeriod",
            "value": "ALARM",
            "timestamp": "2022-07-14T13:59:46.425+0000"
        },
        "configuration": {
            "alarmRule": "ALARM(ServerCpuTooHigh) OR
 ALARM(TotalNetworkTrafficTooHigh)",
            "actionsSuppressor": "ServiceMaintenanceAlarm",
            "actionsSuppressorWaitPeriod": 120,
            "actionsSuppressorExtensionPeriod": 360,
            "alarmName": "ServiceAggregatedAlarm",
            "actionsEnabled": true,
            "timestamp": "2022-07-14T13:59:56.290+0000",
            "okActions": [],
            "alarmActions": [],
            "insufficientDataActions": []
        }
    }
}
```

Start an investigation in Amazon Q operational investigations from an alarm

You can start an investigation in Amazon Q operational investigations from the current state of a CloudWatch alarm, or from any point in the last two weeks of a CloudWatch alarm's history.

For more information about Amazon Q operational investigations, see [Amazon Q Developer operational investigations (Preview)](#).

To start an investigation from a CloudWatch alarm

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the left navigation pane, choose **Alarms**, **All alarms**.
3. Choose the name of the alarm.
4. Choose the time period in the alarm history that you want to investigate.
5. Choose **Investigate**, **Start new investigation**.
6. For **New investigation title**, enter a name for the investigation. Then choose **Start investigation**.

   The Amazon Q investigations assistant starts. It scans your telemetry data to find data that might be associated with this situation.
7. In the CloudWatch console's navigation pane, choose **Investigations**, then choose the name of the investigation that you just started.

   The **Findings** section displays a natural-language summary of the alarm's status and the reason that it was triggered.
8. (Optional) In the graph of the alarm, you can right-click and then choose to deep-dive into the alarm or the metric that it watches.
9. On the right side of the screen, choose the **Suggestions** tab.

   You see a list of other telemetry that Amazon Q operational investigations has discovered and that might be relevant to the investigation. These findings can include other metrics and CloudWatch Logs Insights query results. Amazon Q operational investigations ran these queries based on the alarm.

   - For each finding, you can choose **Add to findings** or **Discard**.

When you choose **Add to findings**, the telemetry is added to the **Findings** section, and Amazon Q operational investigations uses this information to direct it's further scanning and suggestions.

- For a CloudWatch Logs Insights query result, to change or edit the query and re-run it, you can open the context (right-click) menu for by the results, and then choose **Open in Logs Insights**. For more information, see [Analyzing log data with CloudWatch Logs Insights](#).

  If you want to run a different query, when you get to the Logs Insights page you can choose to use query assist to be able to use natural language to form a query. For more information, see [Use natural language to generate and update CloudWatch Logs Insights queries](#).
- (Optional) If you know of telemetry in another Amazon service that might apply to this investigation, you can go to that service's console and add the telemetry to the investigation. For example, to add a Lambda metric to the investigation, you can do the following:

  1. Open the Lambda console.
  2. In the **Monitor** section, find the metric.
  3. Open the context menu for the metric, choose **Investigate**, **Add to investigation** Then, in the **Investigate** pane, select the name of the investigation.

10. Amazon Q might also add hypotheses to the list in the **Suggestions** tab. These hypotheses are generated by the investigation in natural language.

    For each hyypotheses, you can choose **Add to findings** or **Discard**.
11. When you think you have completed the investigation and found the root cause of the issue, you can choose the **Overview** tab and then choose **Investigation summary**. Amazon Q investigations then creates a natural-language summary of the important findings and hypotheses from the investigation.

Managing alarms

Edit or delete a CloudWatch alarm

You can edit or delete an existing alarm.

You can't change the name of an existing alarm. You can copy an alarm and give the new alarm a different name. To copy an alarm, select the check box next to the alarm name in the alarm list and choose **Action**, **Copy**.

To edit an alarm

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Alarms**, **All Alarms**.
3. Choose the name of the alarm.
4. To add or remove tags, choose the **Tags** tab and then choose **Manage tags**.
5. To edit other parts of the alarm, choose **Actions**, **Edit**.

   The **Specify metric and conditions** page appears, showing a graph and other information about the metric and statistic that you selected.
6. To change the metric, choose **Edit**, choose the **All metrics** tab, and do one of the following:
   - Choose the service namespace that contains the metric that you want. Continue choosing options as they appear to narrow the choices. When a list of metrics appears, select the check box next to the metric that you want.
   - In the search box, enter the name of a metric, dimension, or resource ID and press Enter. Then choose one of the results and continue until a list of metrics appears. Select the check box next to the metric that you want.

   Choose **Select metric**.
7. To change other aspects of the alarm, choose the appropriate options. To change how many data points must be breaching for the alarm to go into `ALARM` state or to change how missing data is treated, choose **Additional configuration**.
8. Choose **Next**.
9. Under **Notification**, **Auto Scaling action**, and **EC2 action**, optionally edit the actions taken when the alarm is triggered. Then choose **Next**.
10. Optionally change the alarm description.

    You can't change the name of an existing alarm. You can copy an alarm and give the new alarm a different name. To copy an alarm, select the check box next to the alarm name in the alarm list and choose **Action**, **Copy**.
11. Choose **Next**.

12. Under **Preview and create**, confirm that the information and conditions are what you want, then choose **Update alarm**.

To update an email notification list that was created using the Amazon SNS console

1. Open the Amazon SNS console at https://console.amazonaws.cn/sns/v3/home.
2. In the navigation pane, choose **Topics** and then select the ARN for your notification list (topic).
3. Do one of the following:
    - To add an email address, choose **Create subscription**. For **Protocol**, choose **Email**. For **Endpoint**, enter the email address of the new recipient. Choose **Create subscription**.
    - To remove an email address, choose the **Subscription ID**. Choose **Other subscription actions**, **Delete subscriptions**.
4. Choose **Publish to topic**.

To delete an alarm

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Alarms**.
3. Select the check box to the left of the name of the alarm, and choose **Actions**, **Delete**.
4. Choose **Delete**.

Hide Auto Scaling alarms

When you view your alarms in the Amazon Web Services Management Console, you can hide the alarms related to both Amazon EC2 Auto Scaling and Application Auto Scaling. This feature is available only in the Amazon Web Services Management Console.

To temporarily hide Auto Scaling alarms

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Alarms**, **All alarms**, and select **Hide Auto Scaling alarms**.

Alarm use cases and examples

The following sections provides examples and tutorials for alarms for common use cases.

Topics

Create a billing alarm to monitor your estimated Amazon charges

You can monitor your estimated Amazon charges by using Amazon CloudWatch. When you enable the monitoring of estimated charges for your Amazon account, the estimated charges are calculated and sent several times daily to CloudWatch as metric data.

The alarm triggers when your account billing exceeds the threshold you specify. It triggers only when the current billing exceeds the threshold. It doesn't use projections based on your usage so far in the month.

If you create a billing alarm at a time when your charges have already exceeded the threshold, the alarm goes to the `ALARM` state immediately.

> **Note**
>
> For information about analyzing CloudWatch charges that you have already been billed for, see Analyzing, optimizing, and reducing CloudWatch costs.

Tasks

Enabling billing alerts

Before you can create an alarm for your estimated charges, you must enable billing alerts, so that you can monitor your estimated Amazon charges and create an alarm using billing metric data. After you enable billing alerts, you can't disable data collection, but you can delete any billing alarms that you created.

After you enable billing alerts for the first time, it takes about 15 minutes before you can view billing data and set billing alarms.

Requirements

- You must be signed in using account root user credentials or as an IAM user that has been given permission to view billing information.
- For consolidated billing accounts, billing data for each linked account can be found by logging in as the paying account. You can view billing data for total estimated charges and estimated charges by service for each linked account, in addition to the consolidated account.
- In a consolidated billing account, member linked account metrics are captured only if the payer account enables the **Receive Billing Alerts** preference. If you change which account is your management/payer account, you must enable the billing alerts in the new management/payer account.
- The account must not be part of the Amazon Partner Network (APN) because billing metrics are not published to CloudWatch for APN accounts. For more information, see Amazon Partner Network.

To enable the monitoring of estimated charges

1. Open the Amazon Billing and Cost Management console at https://console.amazonaws.cn/costmanagement/.
2. In the navigation pane, choose **Billing Preferences**.
3. By **Alert preferences** choose **Edit**.
4. Choose **Receive CloudWatch Billing Alerts**.
5. Choose **Save preferences**.

Create a billing alarm

> **Important**
>
> Before you create a billing alarm, you must set your Region to US East (N. Virginia). Billing metric data is stored in this Region and represents worldwide charges. You also must enable billing alerts for your account or in the management/payer account (if you are using consolidated billing). For more information, see Enabling billing alerts.

In this procedure, you create an alarm that sends a notification when your estimated charges for Amazon exceed a defined threshold.

To create a billing alarm using the CloudWatch console

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Alarms**, and then choose **All alarms**.
3. Choose **Create alarm**.
4. Choose **Select metric**. In **Amazon Namespaces**, choose **Billing**, and then choose **Total Estimated Charge**.

   > **Note**
   >
   > If you don't see the **Billing/Total Estimated Charge** metric, enable billing alerts, and change your Region to US East (N. Virginia). For more information, see Enabling billing alerts.

5. Select the box for the **EstimatedCharges** metric, and then choose **Select metric**.
6. For **Statistic**, choose **Maximum**.
7. For **Period**, choose **6 hours**.
8. For **Threshold type**, choose **Static**.
9. For **Whenever EstimatedCharges is . . .**, choose **Greater**.
10. For **than . . .**, define the value that you want to cause your alarm to trigger. For example, **200** USD.

    The **EstimatedCharges** metric values are only in US dollars (USD), and the currency conversion is provided by Amazon Services LLC. For more information, see What is Amazon Billing?.

> **Note**
>
> After you define a threshold value, the preview graph displays your estimated charges for the current month.

11. Choose **Additional Configuration** and do the following:

    - For **Datapoints to alarm**, specify **1 out of 1**.
    - For **Missing data treatment**, choose **Treat missing data as missing**.

12. Choose **Next**.
13. Under **Notification**, ensure that **In alarm** is selected. Then specify an Amazon SNS topic to be notified when your alarm is in the `ALARM` state. The Amazon SNS topic can include your email address so that you receive email when the billing amount crosses the threshold that you specified.

    You can select an existing Amazon SNS topic, create a new Amazon SNS topic, or use a topic ARN to notify other account. If you want your alarm to send multiple notifications for the same alarm state or for different alarm states, choose **Add notification**.

14. Choose **Next**.
15. Under **Name and description**, enter a name for your alarm. The name must contain only UTF-8 characters, and can't contain ASCII control characters.

    - (Optional) Enter a description of your alarm. The description can include markdown formatting, which is displayed only in the alarm **Details** tab in the CloudWatch console. The markdown can be useful to add links to runbooks or other internal resources.

16. Choose **Next**.
17. Under **Preview and create**, make sure that your configuration is correct, and then choose **Create alarm**.

Deleting a billing alarm

You can delete your billing alarm when you no longer need it.

To delete a billing alarm

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.

2. In the navigation pane, choose **Alarms**, **All alarms**.
3. Select the check box next to the alarm and choose **Actions**, **Delete**.
4. When prompted for confirmation, choose **Yes, Delete**.

Create a CPU usage alarm

You can create an CloudWatch alarm that sends a notification using Amazon SNS when the alarm changes state from `OK` to `ALARM`.

The alarm changes to the `ALARM` state when the average CPU use of an EC2 instance exceeds a specified threshold for consecutive specified periods.

Setting up a CPU usage alarm using the Amazon Web Services Management Console

Use these steps to use the Amazon Web Services Management Console to create a CPU usage alarm.

To create an alarm based on CPU usage

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Alarms**, **All Alarms**.
3. Choose **Create alarm**.
4. Choose **Select metric**.
5. In the **All metrics** tab, choose **EC2 metrics**.
6. Choose a metric category (for example, **Per-Instance Metrics**).
7. Find the row with the instance that you want listed in the **InstanceId** column and **CPUUtilization** in the **Metric Name** column. Select the check box next to this row, and choose **Select metric**.
8. Under **Specify metric and conditions**, for **Statistic** choose **Average**, choose one of the predefined percentiles, or specify a custom percentile (for example, **`p95.45`**).
9. Choose a period (for example, **`5 minutes`**).
10. Under **Conditions**, specify the following:

    a. For **Threshold type**, choose **Static**.

b. For **Whenever CPUUtilization is**, specify **Greater**. Under **than...**, specify the threshold that is to trigger the alarm to go to ALARM state if the CPU utilization exceeds this percentage. For example, 70.

c. Choose **Additional configuration**. For **Datapoints to alarm**, specify how many evaluation periods (data points) must be in the `ALARM` state to trigger the alarm. If the two values here match, you create an alarm that goes to `ALARM` state if that many consecutive periods are breaching.

   To create an M out of N alarm, specify a lower number for the first value than you specify for the second value. For more information, see Evaluating an alarm.

d. For **Missing data treatment**, choose how to have the alarm behave when some data points are missing. For more information, see Configuring how CloudWatch alarms treat missing data.

e. If the alarm uses a percentile as the monitored statistic, a **Percentiles with low samples** box appears. Use it to choose whether to evaluate or ignore cases with low sample rates. If you choose **ignore (maintain alarm state)**, the current alarm state is always maintained when the sample size is too low. For more information, see Percentile-based CloudWatch alarms and low data samples.

11. Choose **Next**.

12. Under **Notification**, choose **In alarm** and select an SNS topic to notify when the alarm is in `ALARM` state

    To have the alarm send multiple notifications for the same alarm state or for different alarm states, choose **Add notification**.

    To have the alarm not send notifications, choose **Remove**.

13. When finished, choose **Next**.

14. Enter a name and description for the alarm. Then choose **Next**.

    The name must contain only UTF-8 characters, and can't contain ASCII control characters. The description can include markdown formatting, which is displayed only in the alarm **Details** tab in the CloudWatch console. The markdown can be useful to add links to runbooks or other internal resources.

15. Under **Preview and create**, confirm that the information and conditions are what you want, then choose **Create alarm**.

Setting up a CPU usage alarm using the Amazon CLI

Use these steps to use the Amazon CLI to create a CPU usage alarm.

To create an alarm based on CPU usage

1. Set up an SNS topic. For more information, see Setting up Amazon SNS notifications.
2. Create an alarm using the put-metric-alarm command as follows.

```
aws cloudwatch put-metric-alarm --alarm-name cpu-mon --alarm-description "Alarm
 when CPU exceeds 70%" --metric-name CPUUtilization --namespace AWS/EC2 --statistic
 Average --period 300 --threshold 70 --comparison-operator GreaterThanThreshold --
dimensions  Name=InstanceId,Value=i-12345678 --evaluation-periods 2 --alarm-actions
 arn:aws:sns:us-east-1:111122223333:my-topic --unit Percent
```

3. Test the alarm by forcing an alarm state change using the set-alarm-state command.

   a. Change the alarm state from `INSUFFICIENT_DATA` to `OK`.

   ```
   aws cloudwatch set-alarm-state --alarm-name cpu-mon --state-reason
    "initializing" --state-value OK
   ```

   b. Change the alarm state from `OK` to `ALARM`.

   ```
   aws cloudwatch set-alarm-state --alarm-name cpu-mon --state-reason
    "initializing" --state-value ALARM
   ```

   c. Check that you have received a notification about the alarm.

Create a load balancer latency alarm that sends email

You can set up an Amazon SNS notification and configure an alarm that monitors latency exceeding 100 ms for your Classic Load Balancer.

Setting up a latency alarm using the Amazon Web Services Management Console

Use these steps to use the Amazon Web Services Management Console to create a load balancer latency alarm.

To create a load balancer latency alarm

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Alarms**, **All Alarms**.
3. Choose **Create alarm**.
4. Under **CloudWatch Metrics by Category**, choose the **ELB Metrics** category.
5. Select the row with the Classic Load Balancer and the **Latency** metric.
6. For the statistic, choose **Average**, choose one of the predefined percentiles, or specify a custom percentile (for example, **`p95.45`**).
7. For the period, choose **1 Minute**.
8. Choose **Next**.
9. Under **Alarm Threshold**, enter a unique name for the alarm (for example, **`myHighCpuAlarm`**) and a description of the alarm (for example, **`Alarm when Latency exceeds 100s`**). Alarm names must contain only UTF-8 characters, and can't contain ASCII control characters

   The name must contain only UTF-8 characters, and can't contain ASCII control characters. The description can include markdown formatting, which is displayed only in the alarm **Details** tab in the CloudWatch console. The markdown can be useful to add links to runbooks or other internal resources.
10. Under **Whenever**, for **is**, choose **>** and enter **`0.1`**. For **for**, enter **`3`**.
11. Under **Additional settings**, for **Treat missing data as**, choose **ignore (maintain alarm state)** so that missing data points don't trigger alarm state changes.

    For **Percentiles with low samples**, choose **ignore (maintain the alarm state)** so that the alarm evaluates only situations with adequate numbers of data samples.
12. Under **Actions**, for **Whenever this alarm**, choose **State is ALARM**. For **Send notification to**, choose an existing SNS topic or create a new one.

    To create an SNS topic, choose **New list**. For **Send notification to**, enter a name for the SNS topic (for example, **`myHighCpuAlarm`**), and for **Email list**, enter a comma-separated list of email addresses to be notified when the alarm changes to the `ALARM` state. Each email address is sent a topic subscription confirmation email. You must confirm the subscription before notifications can be sent.
13. Choose **Create Alarm**.

Setting up a latency alarm using the Amazon CLI

Use these steps to use the Amazon CLI to create a load balancer latency alarm.

To create a load balancer latency alarm

1. Set up an SNS topic. For more information, see Setting up Amazon SNS notifications.
2. Create the alarm using the put-metric-alarm command as follows:

```
aws cloudwatch put-metric-alarm --alarm-name lb-mon --alarm-description "Alarm
 when Latency exceeds 100s" --metric-name Latency --namespace AWS/ELB --statistic
 Average --period 60 --threshold 100 --comparison-operator GreaterThanThreshold --
dimensions Name=LoadBalancerName,Value=my-server --evaluation-periods 3 --alarm-
actions arn:aws:sns:us-east-1:111122223333:my-topic --unit Seconds
```

3. Test the alarm by forcing an alarm state change using the set-alarm-state command.

   a. Change the alarm state from `INSUFFICIENT_DATA` to `OK`.

   ```
   aws cloudwatch set-alarm-state --alarm-name lb-mon --state-reason
    "initializing" --state-value OK
   ```

   b. Change the alarm state from `OK` to `ALARM`.

   ```
   aws cloudwatch set-alarm-state --alarm-name lb-mon --state-reason
    "initializing" --state-value ALARM
   ```

   c. Check that you have received an email notification about the alarm.

Create a storage throughput alarm that sends email

You can set up an SNS notification and configure an alarm that is triggered when Amazon EBS exceeds 100 MB throughput.

Setting up a storage throughput alarm using the Amazon Web Services Management Console

Use these steps to use the Amazon Web Services Management Console to create an alarm based on Amazon EBS throughput.

To create a storage throughput alarm

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Alarms**, **All Alarms**.
3. Choose **Create alarm**.
4. Under **EBS Metrics**, choose a metric category.
5. Select the row with the volume and the **VolumeWriteBytes** metric.
6. For the statistic, choose **Average**. For the period, choose **5 Minutes**. Choose **Next**.
7. Under **Alarm Threshold**, enter a unique name for the alarm (for example, **`myHighWriteAlarm`**) and a description of the alarm (for example, **`VolumeWriteBytes exceeds 100,000 KiB/s`**). The name must contain only UTF-8 characters, and can't contain ASCII control characters. The description can include markdown formatting, which is displayed only in the alarm **Details** tab in the CloudWatch console. The markdown can be useful to add links to runbooks or other internal resources.
8. Under **Whenever**, for **is**, choose **>** and enter **`100000`**. For **for**, enter **15** consecutive periods.

   A graphical representation of the threshold is shown under **Alarm Preview**.
9. Under **Additional settings**, for **Treat missing data as**, choose **ignore (maintain alarm state)** so that missing data points don't trigger alarm state changes.
10. Under **Actions**, for **Whenever this alarm**, choose **State is ALARM**. For **Send notification to**, choose an existing SNS topic or create one.

    To create an SNS topic, choose **New list**. For **Send notification to**, enter a name for the SNS topic (for example, **`myHighCpuAlarm`**), and for **Email list**, enter a comma-separated list of email addresses to be notified when the alarm changes to the `ALARM` state. Each email address is sent a topic subscription confirmation email. You must confirm the subscription before notifications can be sent to an email address.
11. Choose **Create Alarm**.

Setting up a storage throughput alarm using the Amazon CLI

Use these steps to use the Amazon CLI to create an alarm based on Amazon EBS throughput.

To create a storage throughput alarm

1. Create an SNS topic. For more information, see [Setting up Amazon SNS notifications](#).

2. Create the alarm.

```
aws cloudwatch put-metric-alarm --alarm-name ebs-mon --alarm-description "Alarm
 when EBS volume exceeds 100MB throughput" --metric-name VolumeReadBytes --
namespace AWS/EBS --statistic Average --period 300 --threshold 100000000 --
comparison-operator GreaterThanThreshold --dimensions Name=VolumeId,Value=my-
volume-id --evaluation-periods 3 --alarm-actions arn:aws:sns:us-
east-1:111122223333:my-alarm-topic --insufficient-data-actions arn:aws:sns:us-
east-1:111122223333:my-insufficient-data-topic
```

3. Test the alarm by forcing an alarm state change using the set-alarm-state command.

   a. Change the alarm state from INSUFFICIENT_DATA to OK.

   ```
   aws cloudwatch set-alarm-state --alarm-name ebs-mon --state-reason
    "initializing" --state-value OK
   ```

   b. Change the alarm state from OK to ALARM.

   ```
   aws cloudwatch set-alarm-state --alarm-name ebs-mon --state-reason
    "initializing" --state-value ALARM
   ```

   c. Change the alarm state from ALARM to INSUFFICIENT_DATA.

   ```
   aws cloudwatch set-alarm-state --alarm-name ebs-mon --state-reason
    "initializing" --state-value INSUFFICIENT_DATA
   ```

   d. Check that you have received an email notification about the alarm.

Create an alarm on Performance Insights counter metrics from an Amazon database

CloudWatch includes a **DB_PERF_INSIGHTS** metric math function which you can use to bring Performance Insights counter metrics into CloudWatch from Amazon Relational Database Service and Amazon DocumentDB (with MongoDB compatibility). **DB_PERF_INSIGHTS** also brings in the DBLoad metric at sub-minute intervals. You can set CloudWatch alarms on these metrics.

For more information about Amazon RDS Performance Insights, see Monitoring DB load with Performance Insights on Amazon RDS.

For more information about Amazon DocumentDB Performance Insights, see [Monitoring with Performance Insights](#).

Anomaly detection is not supported for alarms based on the **DB_PERF_INSIGHTS** function.

> **Note**
>
> High-resolution metrics with sub-minute granularity retrieved by **DB_PERF_INSIGHTS** are only applicable to the **DBLoad** metric, or for operating system metrics if you have enabled Enhanced Monitoring at a higher resolution. For more information about Amazon RDS enhanced monitoring, see [Monitoring OS metrics with Enhanced Monitoring.](#).
> You can create a high-resolution alarm using the **DB_PERF_INSIGHTS** function. The maximum evaluation range for a high-resolution alarm is three hours. You can use the CloudWatch console to graph metrics retrieved with the **DB_PERF_INSIGHTS** function for any time range.

To create an alarm that's based on Performance Insights metrics

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Alarms**, and then choose **All alarms**.
3. Choose **Create alarm**.
4. Choose **Select Metric**.
5. Choose the **Add math** dropdown, and then select **All functions**, **DB_PERF_INSIGHTS** from the list.

   After you choose **DB_PERF_INSIGHTS**, a math expression box appears where you apply or edit math expressions.
6. In the math expression box, enter your **DB_PERF_INSIGHTS** math expression, and then choose **Apply**.

   For example, `DB_PERF_INSIGHTS(‘RDS’, ‘db-ABCDEFGHIJKLMNOPQRSTUVWXY1’, ‘os.cpuUtilization.user.avg’)`

   > **Important**
   >
   > When you use the **DB_PERF_INSIGHTS** math expression, you must specify the Unique Database Resource ID of the database. This is different than the database identifier. To

find the database resource ID in the Amazon RDS console, choose the DB instance to see its details. Then choose the **Configuration** tab. The **Resource ID** is displayed in the **Configuration** section.

For information about the **DB_PERF_INSIGHTS** function and other functions that are available for metric math, see [Metric math syntax and functions](#).

7. Choose **Select metric**.

   The **Specify metric and conditions** page appears, showing a graph and other information about the math expression that you have selected.

8. For **Whenever *`expression`* is**, specify whether the expression must be greater than, less than, or equal to the threshold. Under **than...**, specify the threshold value.

9. Choose **Additional configuration**. For **Datapoints to alarm**, specify how many evaluation periods (data points) must be in the `ALARM` state to trigger the alarm. If the two values here match, you create an alarm that goes to `ALARM` state if that many consecutive periods are breaching.

   To create an M out of N alarm, specify a lower number for the first value than you specify for the second value. For more information, see [Evaluating an alarm](#).

10. For **Missing data treatment**, choose how to have the alarm behave when some data points are missing. For more information, see [Configuring how CloudWatch alarms treat missing data](#).

11. Choose **Next**.

12. Under **Notification**, select an SNS topic to notify when the alarm is in `ALARM` state, `OK` state, or `INSUFFICIENT_DATA` state.

    To have the alarm send multiple notifications for the same alarm state or for different alarm states, choose **Add notification**.

    To have the alarm not send notifications, choose **Remove**.

13. To have the alarm perform Auto Scaling, EC2, Lambda, or Systems Manager actions, choose the appropriate button and choose the alarm state and action to perform. If you choose a Lambda function as an alarm action, you specify the function name or ARN, and you can optionally choose a specific version of the function.

Alarms can perform Systems Manager actions only when they go into ALARM state. For more information about Systems Manager actions, see see Configuring CloudWatch to create OpsItems from alarms and Incident creation.

> **Note**
>
> To create an alarm that performs an SSM Incident Manager action, you must have certain permissions. For more information, see Identity-based policy examples for Amazon Systems Manager Incident Manager.

14. When finished, choose **Next**.
15. Enter a name and description for the alarm. Then choose **Next**.

    The name must contain only UTF-8 characters, and can't contain ASCII control characters. The description can include markdown formatting, which is displayed only in the alarm **Details** tab in the CloudWatch console. The markdown can be useful to add links to runbooks or other internal resources.
16. Under **Preview and create**, confirm that the information and conditions are what you want, then choose **Create alarm**.

Create alarms to stop, terminate, reboot, or recover an EC2 instance

Using Amazon CloudWatch alarm actions, you can create alarms that automatically stop, terminate, reboot, or recover your EC2 instances. You can use the stop or terminate actions to help you save money when you no longer need an instance to be running. You can use the reboot and recover actions to automatically reboot those instances or recover them onto new hardware if a system impairment occurs.

There are a number of scenarios in which you might want to automatically stop or terminate your instance. For example, you might have instances dedicated to batch payroll processing jobs or scientific computing tasks that run for a period of time and then complete their work. Rather than letting those instances sit idle (and accrue charges), you can stop or terminate them, which helps you to save money. The main difference between using the stop and the terminate alarm actions is that you can easily restart a stopped instance if you need to run it again later. You can also keep the same instance ID and root volume. However, you cannot restart a terminated instance. Instead, you must launch a new instance.

You can add the stop, terminate, or reboot, actions to any alarm that is set on an Amazon EC2 per-instance metric, including basic and detailed monitoring metrics provided by Amazon CloudWatch (in the AWS/EC2 namespace), in addition to any custom metrics that include the "InstanceId=" dimension, as long as the InstanceId value refers to a valid running Amazon EC2 instance. You can also add the recover action to alarms that is set on any Amazon EC2 per-instance metric except for `StatusCheckFailed_Instance`.

> **Important**
>
> Alarms configured on Amazon EC2 metrics can temporarily enter the INSUFFICIENT_DATA state if there are missing metric data points. This is rare, but can happen when the metric reporting is interrupted, even when the Amazon EC2 instance is healthy. For alarms on Amazon EC2 metrics that are configured to take stop, terminate, reboot, or recover actions, we recommend that you configure those alarms to treat missing data as `missing`, and to have these alarms trigger only when in the ALARM state.
> For more information about how you can configure CloudWatch to act on missing metrics that have alarms set on them, see Configuring how CloudWatch alarms treat missing data.

To set up a CloudWatch alarm action that can reboot, stop, or terminate an instance, you must use a service-linked IAM role, *AWSServiceRoleForCloudWatchEvents*. The AWSServiceRoleForCloudWatchEvents IAM role enables Amazon to perform alarm actions on your behalf.

To create the service-linked role for CloudWatch Events, use the following command:

```
aws iam create-service-linked-role --aws-service-name events.amazonaws.com
```

Console support

You can create alarms using the CloudWatch console or the Amazon EC2 console. The procedures in this documentation use the CloudWatch console. For procedures that use the Amazon EC2 console, see Create alarms that stop, terminate, reboot, or rcover an instance in the *Amazon EC2 User Guide*.

Permissions

If you are using an Amazon Identity and Access Management (IAM) account to create or modify an alarm that performs EC2 actions or Systems Manager OpsItem actions, you must have the `iam:CreateServiceLinkedRole` permission.

Contents

Adding stop actions to Amazon CloudWatch alarms

You can create an alarm that stops an Amazon EC2 instance when a certain threshold has been met. For example, you may run development or test instances and occasionally forget to shut them off. You can create an alarm that is triggered when the average CPU utilization percentage has been lower than 10 percent for 24 hours, signaling that it is idle and no longer in use. You can adjust the threshold, duration, and period to suit your needs, plus you can add an SNS notification, so that you will receive an email when the alarm is triggered.

Amazon EC2 instances that use an Amazon Elastic Block Store volume as the root device can be stopped or terminated, whereas instances that use the instance store as the root device can only be terminated.

To create an alarm to stop an idle instance using the Amazon CloudWatch console

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Alarms**, **All alarms**.
3. Choose **Create alarm**.
4. Choose **Select Metric**.
5. For **Amazon namespaces**, choose **EC2**.
6. Do the following:

    a. Choose **Per-Instance Metrics**.

    b. Select the check box in the row with the correct instance and the **CPUUtilization** metric.

    c. Choose the **Graphed metrics** tab.

   d. For the statistic, choose **Average**.
   e. Choose a period (for example, **1 Hour**).
   f. Choose **Select metric**.
7. Under **Conditions**, do the following:
   a. Choose **Static**.
   b. Under **Whenever CPUUtilization is**, choose **Lower**.
   c. For **than**, type **10**.
   d. Choose **Next**.
   e. Under **Notification**, for **Send notification to**, choose an existing SNS topic or create a new one.

      To create an SNS topic, choose **New list**. For **Send notification to**, type a name for the SNS topic (for example, Stop_EC2_Instance). For **Email list**, type a comma-separated list of email addresses to be notified when the alarm changes to the `ALARM` state. Each email address is sent a topic subscription confirmation email. You must confirm the subscription before notifications can be sent to an email address.
   f. Choose **Add EC2 Action**.
   g. For **Alarm state trigger**, choose **In alarm**. For **Take the following action**, choose **Stop this instance**.
   h. Choose **Next**.
   i. Enter a name and description for the alarm. The name must contain only ASCII characters. Then choose **Next**.
   j. Under **Preview and create**, confirm that the information and conditions are what you want, then choose **Create alarm**.

Adding terminate actions to Amazon CloudWatch alarms

You can create an alarm that terminates an EC2 instance automatically when a certain threshold has been met (as long as termination protection is not enabled for the instance). For example, you might want to terminate an instance when it has completed its work, and you don't need the instance again. If you might want to use the instance later, you should stop the instance instead of terminating it. For information about enabling and disabling termination protection for an instance, see Enabling Termination Protection for an Instance in the *Amazon EC2 User Guide*.

To create an alarm to terminate an idle instance using the Amazon CloudWatch console

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Alarms**, **Create Alarm**.
3. For the **Select Metric** step, do the following:

   a. Under **EC2 Metrics**, choose **Per-Instance Metrics**.

   b. Select the row with the instance and the **CPUUtilization** metric.

   c. For the statistic, choose **Average**.

   d. Choose a period (for example, **`1 Hour`**).

   e. Choose **Next**.
4. For the **Define Alarm** step, do the following:

   a. Under **Alarm Threshold**, type a unique name for the alarm (for example, Terminate EC2 instance) and a description of the alarm (for example, Terminate EC2 instance when CPU is idle for too long). Alarm names must contain only ASCII characters.

   b. Under **Whenever**, for **is**, choose **<** and type **`10`**. For **for**, type **24** consecutive periods.

      A graphical representation of the threshold is shown under **Alarm Preview**.

   c. Under **Notification**, for **Send notification to**, choose an existing SNS topic or create a new one.

      To create an SNS topic, choose **New list**. For **Send notification to**, type a name for the SNS topic (for example, Terminate_EC2_Instance). For **Email list**, type a comma-separated list of email addresses to be notified when the alarm changes to the `ALARM` state. Each email address is sent a topic subscription confirmation email. You must confirm the subscription before notifications can be sent to an email address.

   d. Choose **EC2 Action**.

   e. For **Whenever this alarm**, choose **State is ALARM**. For **Take this action**, choose **Terminate this instance**.

   f. Choose **Create Alarm**.

Adding reboot actions to Amazon CloudWatch alarms

You can create an Amazon CloudWatch alarm that monitors an Amazon EC2 instance and automatically reboots the instance. The reboot alarm action is recommended for Instance Health

Check failures (as opposed to the recover alarm action, which is suited for System Health Check failures). An instance reboot is equivalent to an operating system reboot. In most cases, it takes only a few minutes to reboot your instance. When you reboot an instance, it remains on the same physical host, so your instance keeps its public DNS name, private IP address, and any data on its instance store volumes.

Rebooting an instance doesn't start a new instance billing hour, unlike stopping and restarting your instance. For more information about rebooting an instance, see Reboot Your Instance in the *Amazon EC2 User Guide*.

> **Important**
>
> To avoid a race condition between the reboot and recover actions, avoid setting the same evaluation period for both a reboot alarm and a recover alarm. We recommend that you set reboot alarms to three evaluation periods of one minute each.

To create an alarm to reboot an instance using the Amazon CloudWatch console

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Alarms**, **Create Alarm**.
3. For the **Select Metric** step, do the following:
   a. Under **EC2 Metrics**, choose **Per-Instance Metrics**.
   b. Select the row with the instance and the **StatusCheckFailed_Instance** metric.
   c. For the statistic, choose **Minimum**.
   d. Choose a period (for example, **`1 Minute`**).
   e. Choose **Next**.
4. For the **Define Alarm** step, do the following:
   a. Under **Alarm Threshold**, type a unique name for the alarm (for example, Reboot EC2 instance) and a description of the alarm (for example, Reboot EC2 instance when health checks fail). Alarm names must contain only ASCII characters.
   b. Under **Whenever**, for **is**, choose **>** and type **`0`**. For **for**, type **3** consecutive periods.

      A graphical representation of the threshold is shown under **Alarm Preview**.

c. Under **Notification**, for **Send notification to**, choose an existing SNS topic or create a new one.

   To create an SNS topic, choose **New list**. For **Send notification to**, type a name for the SNS topic (for example, Reboot_EC2_Instance). For **Email list**, type a comma-separated list of email addresses to be notified when the alarm changes to the `ALARM` state. Each email address is sent a topic subscription confirmation email. You must confirm the subscription before notifications can be sent to an email address.

d. Choose **EC2 Action**.

e. For **Whenever this alarm**, choose **State is ALARM**. For **Take this action**, choose **Reboot this instance**.

f. Choose **Create Alarm**.

Adding recover actions to Amazon CloudWatch alarms

You can create an Amazon CloudWatch alarm that monitors an Amazon EC2 instance and automatically recovers the instance if it becomes impaired due to an underlying hardware failure or a problem that requires Amazon involvement to repair. Terminated instances cannot be recovered. A recovered instance is identical to the original instance, including the instance ID, private IP addresses, Elastic IP addresses, and all instance metadata.

When the `StatusCheckFailed_System` alarm is triggered, and the recover action is initiated, you will be notified by the Amazon SNS topic that you chose when you created the alarm and associated the recover action. During instance recovery, the instance is migrated during an instance reboot, and any data that is in-memory is lost. When the process is complete, information is published to the SNS topic you've configured for the alarm. Anyone who is subscribed to this SNS topic will receive an email notification that includes the status of the recovery attempt and any further instructions. You will notice an instance reboot on the recovered instance.

The recover action can be used only with `StatusCheckFailed_System`, not with `StatusCheckFailed_Instance`.

Examples of problems that cause system status checks to fail include:

- Loss of network connectivity
- Loss of system power
- Software issues on the physical host

- Hardware issues on the physical host that impact network reachability

The recover action is supported only on some instance types. For more information about supported instance types and other requirements, see [Recover your instance] and [Requirements].

> **Important**
>
> To avoid a race condition between the reboot and recover actions, avoid setting the same evaluation period for both a reboot alarm and a recover alarm. We recommend that you set recover alarms to two evaluation periods of one minute each and reboot alarms to three evaluation periods of one minute each.

To create an alarm to recover an instance using the Amazon CloudWatch console

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Alarms**, **All alarms**.
3. Choose **Create Alarm**.
4. Choose **Select Metric** and then do the following:

    a. Choose **EC2 Metrics**, **Per-Instance Metrics**.

    b. Select the row with the instance and the **StatusCheckFailed_System** metric, and then choose **Select metric**.

    c. For the statistic, choose **Minimum**.

    d. Choose a period (for example, **`1 Minute`**).

    > **Important**
    >
    > To avoid a race condition between the reboot and recover actions, avoid setting the same evaluation period for both a reboot alarm and a recover alarm. We recommend that you set recover alarms to two evaluation periods of one minute each.

5. For **Conditions**, do the following:

    a. Under **Threshold type**, choose **Static**.

    b. Under **Whenever**, choose **Greater** and enter **`0`** for **than...**.

c. Choose **Additional configuration**, then for **Datapoints to alarm** specify 2 **out of** 2.

6. Choose **Next**.
7. Under **Notification**, do the following:

   a. For **Alarm state trigger**, choose **In alarm**.
   b. For **Send notification to the following SNS topic**, choose an existing SNS topic or create a new one.
   c. Choose **Add EC2 Action**.
   d. For **Alarm state trigger**, choose **In alarm**.
   e. For **Take the following action**, choose **Recover this instance**.
   f. Choose **Next**.
8. For **Alarm name**, type a unique name for the alarm (for example, **`Recover EC2 instance`**) and a description of the alarm (for example, **`Recover EC2 instance when health checks fail`**). Alarm names must contain only ASCII characters.
9. Choose **Next**.
10. Choose **Create Alarm**.

Viewing the history of triggered alarms and actions

You can view alarm and action history in the Amazon CloudWatch console. Amazon CloudWatch keeps the last 30 days of alarm and action history.

To view the history of triggered alarms and actions

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Alarms** and select an alarm.
3. To view the most recent state transition along with the time and metric values, choose **Details**.
4. To view the most recent history entries, choose **History**.

Alarms and tagging

Tags are key-value pairs that can help you organize and categorize your resources. You can also use them to scope user permissions by granting a user permission to access or change only resources with certain tag values. For more general information about tagging resources, see [Tagging your Amazon resources]

The following list explains some details about how tagging works with CloudWatch alarms.

- To be able to set or update tags for a CloudWatch resource, you must be signed on to an account that has the `cloudwatch:TagResource` permission. For example, to create an alarm and set tags for it, you must have the `cloudwatch:TagResource` permission in addition to the `the cloudwatch:PutMetricAlarm` permission. We recommend that you make sure anyone in your organization who will create or update CloudWatch resources has the `cloudwatch:TagResource` permission.
- Tags can be used for tag-based authorization control. For example, IAM user or role permissions can include conditions to limit CloudWatch calls to specific resources based on their tags. However, keep in mind the following
  - Tags with names that start with `aws:` can't be used for tag-based authorization control.
  - Composite alarms do not support tag-based authorization control.

Auditing CloudWatch telemetry configurations

You can use Amazon CloudWatch to discover and understand the state of telemetry configuration for your Amazon resources from a central view in the CloudWatch console. This simplifies the process of auditing your telemetry collection configurations across multiple resource types within an account or across multiple accounts in Organizations. With a consolidated view, you can easily review and manage telemetry settings, helping you to ensure proper monitoring and data collection across your Amazon environment.

CloudWatch can help you identify telemetry configuration for the following types of Amazon resource types:

- Amazon EC2 instances that provide detailed metrics. For more information, see Manage detailed monitoring for your EC2 instances in the Amazon EC2 User Guide.
- Amazon VPC virtual networks that provide flow logs. For more information, see Logging IP traffic using VPC Flow Logs in the Amazon VPC User Guide.
- Lambda functions that provide traces. For more information, see Visualize Lambda function invocations using Amazon X-Ray in the Amazon X-Ray Developer Guide.

To begin auditing your telemetry configurations, you must first turn on the telemetry auditing experience for your Amazon Web Services account or organization. Enabling this feature creates Amazon Config service-linked configuration recorders that discover resources and their associated telemetry configuration metadata. For more information, see Configuration Recorder in the Amazon Config Developer Guide.

> **Note**
>
> Amazon Config periodically takes inventory of, or discovers, all the resources in your account as an anti-entropy behavior, regardless of the resource types in scope for your configuration recorders. The inventory includes deleted resources and resources that Amazon Config is not currently recording. This behavior helps maintain data consistency. This means that although the service-linked configuration recorder for the CloudWatch telemetry auditing feature is configured to record three resource types (Amazon EC2 instances, Amazon EC2 VPC virtual networks, and Lambda functions), you might see describe calls from `ConfigResourceCompositionSession` and `AWSConfig-Describe`

in Amazon CloudTrail. For more information, see Non-recorded Resources in the Amazon Config Developer Guide.

The telemetry auditing experience uses this information and offers visibility into the configuration status, at the resource type level and at more granular telemetry detail levels. You can customize your view of the resources or telemetry details using filters, and modify the telemetry configuration directly from the resource's console page.

Turning on the telemetry auditing experience does not incur any additional costs.

Topics

- Turning on CloudWatch telemetry auditing
- Viewing Amazon resource telemetry in CloudWatch
- Turning off CloudWatch telemetry auditing

Turning on CloudWatch telemetry auditing

Use the CloudWatch console to turn on telemetry auditing for your Amazon Web Services account or organization. For an organization, CloudWatch uses a management account or a delegated administrator account to discover Amazon resources and the telemetry configurations for all of the member accounts in the organization. Turning on the telemetry auditing experience does not incur any additional cost.

Telemetry auditing remains on until you turn it off. For more information, see Turning off CloudWatch telemetry auditing.

Topics

- Auditing telemetry configurations for your organization
- Turning on telemetry auditing for your account
- Deregistering a delegated administrator account
- Turning off trusted access for Organizations

Auditing telemetry configurations for your organization

To turn on telemetry configuration for your organization, you must use a management account or a delegated administrator account. CloudWatch uses this account to discover your organization's Amazon resources and their telemetry configurations.

Before you can turn on telemetry auditing for your organization, you need to turn on trusted access between Amazon Organizations and CloudWatch. When you turn on trusted access, CloudWatch creates a service-linked role named **AWSServiceRoleForObservabilityAdmin** to support resource and telemetry configuration discovery for the organization. The role is created in all member accounts of the organization. For more information about the service-linked role, see Service-linked role permissions for CloudWatch telemetry config. For more information about Amazon Organizations, see Amazon CloudWatch and Amazon Organizations in the Amazon Organizations User Guide.

To use a management account for telemetry configuration, log in with the account, turn on trusted access, and then turn on telemetry auditing. For more information, see Turning on telemetry auditing for your Amazon Organizations.

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pain, choose **Settings**.
3. Choose the **Organizations** tab.
4. In **Organizational Management Settings**, choose **Turn on**. The **Enable trusted access** page appears.
5. To review the role policy, choose **View permission details** and the role policy appears in a window. Choose **Enable trusted access**. The telemetry configuration **Overview** page appears and CloudWatch begins discovering Amazon resources in the organization. As CloudWatch discovers resources, it updates information on the **Overview** page.

> **Note**
>
> The time delay before resources appear on the **Overview** page depends on the number of member accounts and resources in your organization or account.

Registering a delegated administrator account for your organization

A delegated administrator account is a member account that shares administrator access for service-managed permissions. The account that you register as a delegated administrator must be in your organization. A delegated administrator account for your organization can be used outside of CloudWatch, so make sure that you understand this account type before you follow this procedure. For more information, see [Amazon CloudWatch and Amazon Organizations]() in the Amazon Organizations User Guide.

To remove or change the delegated administrator account, deregister the account first. For more information, see [Deregistering a delegated administrator account]().

To register a delegated administrator account

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Settings**.
3. Choose the **Organization** tab.
4. Choose **Register delegated administrator**.
5. On the **Register delegated administrator** page, for **Delegated administrator account ID**, enter the 12-digit account ID for an organization member.
6. Choose **Register delegated administrator**. At the top of the page, a message appears indicating the account was registered successfully. The **Organization Settings** page appears. To see information about the delegated administrator account, hover over the number below **Delegated administrators**.

Turning on telemetry auditing for your Amazon Organizations

Turn on telemetry auditing for your Amazon Organizations to monitor the telemetry for the Amazon resources across all your member accounts. This also turns on the telemetry auditing experience for individual accounts. You can also turn on the telemetry auditing experience for only your account. For more information, see [Turning on telemetry auditing for your account]().

You can turn off trusted access across all your member accounts. For more information, see [Turning off trusted access for Organizations]().

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Telemetry config**.

3. Choose **Turn on**, and then choose the **Organization** tab. The telemetry config **Overview** page appears and CloudWatch begins discovering Amazon resources in your account. As CloudWatch discovers resources, it updates information in the **Overview** page.

> **Note**
>
> The delay before resources appear on the **Overview** page depends on the number of member accounts and resources in your organization or account.

Turning on telemetry auditing for your account

Turn on telemetry auditing for your Amazon Web Services account to monitor telemetry for the Amazon resources in that account. If you have an organization in Amazon Organizations, turn on telemetry configuration for your organization instead. For more information, see [Turning on telemetry auditing for your Amazon Organizations](#).

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Settings**, **Telemetry config**.
3. Choose **Turn on**, then **This account**, if you are using a management account or a delegated administrator account. The telemetry configuration **Overview** page appears and CloudWatch begins discovering Amazon resources in your account. As CloudWatch discovers resources, it updates information on the **Overview** page.

> **Note**
>
> The delay before resources appear on the **Overview** page depends on the number of member accounts and resources in your organization or account.

Deregistering a delegated administrator account

Deregister the delegated administrator account before turning off trusted access for Organizations. You can also deregister a delegated administrator account if it no longer has access to the appropriate Amazon resources for telemetry auditing or to choose a different member account to be the delegated administrator. This account will not be able to perform account management tasks for Organizations. For more information, see [Amazon CloudWatch and Amazon Organizations] in the Amazon Organizations User Guide.

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Settings**.
3. On the **Organization** tab, choose **Deregister**.
4. On the **Deregister delegated administrator** page, choose **Deregister**.

To register an account as a delegated administrator, see Registering a delegated administrator account for your organization.

Turning off trusted access for Organizations

Trusted access extends the functionality of the management account in Amazon Organizations to other Amazon services. When you turn off trusted access, trusted access between your organization and all Amazon services—not just CloudWatch—will stop.

If you no longer want trusted access turned on for your organization, you can turn it off. For more information, see Amazon CloudWatch and Amazon Organizations in the Amazon Organizations User Guide.

> **Note**
>
> Before turning off trusted access for an organization, deregister the delegated administrator account. For more information, see Deregistering a delegated administrator account.

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Settings**.
3. Choose the **Organization** tab.
4. In the **Organizational Management Settings** section, select **Turn off**.

Viewing Amazon resource telemetry in CloudWatch

Telemetry configuration displays Amazon resources in two places: As an overview, on the **Telemetry config** page and in detail, on the **Discovered resources** page.

- **Telemetry config** – This page shows the percentage of resources with telemetry that are configured for each resource type and the number of resources detected as providing metrics, as well as the total number of resources in your account or organization. You can filter the display of resources in the **Telemetry config** page by account ID or by the tags applied to your resources.
- **Discovered resources** – This page shows details about each resource that has been discovered by telemetry configuration, including the resource ID, the type of telemetry each resource is providing, and the time when information about the resource was last refreshed by telemetry configuration. You can filter the display of resource on the **Discovered resources** page by any of the information provided about the resource.

  For each Amazon resource tracked by telemetry configuration, the **Discovered resources** page shows the status of its telemetry by providing the following information:

  - For telemetry types that CloudWatch detects that the resource is providing, the **Discovered resources** page shows **On**.
  - For telemetry types that CloudWatch detects the resource is not providing, the **Discovered resources** page shows **Off**.
  - For telemetry types that are not supported for a resource, the **Discovered resources** page shows **NS**, that is, not supported.

To view resources on the Telemetry config page

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Telemetry config**.
3. The **Telemetry config** page shows the total number of each resource that was discovered by telemetry config, the number of resources providing telemetry, and the percentage of discovered resources that are providing telemetry.
4. To see recent changes to resources, choose **Refresh**.

To view resources on the Discovered resources page

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Telemetry config**.
3. Do one of the following to view all resource types discovered by telemetry configuration or to view one resource type:

   a. To view all resources that have been discovered by telemetry configuration, choose **View resources**. The **Discovered resources** page appears and shows all resources discovered by telemetry configuration.

   b. To view one resource type, at the bottom of the page, choose a type of Amazon resource. The **Discovered resources** page appears. The **Discovered resources** page shows all discovered resources for that resource type.

4. On the **Discovered resources** page, do one or more of the following:

   - To view information about the resource or to change its telemetry settings, choose a resource ID. The console page for the resource appears.

     > **Note**
     >
     > You can only view a resource on its console page if the resource belongs to your account. To determine if the resource belongs to your account, check the **Amazon Web Services account** column. If the **Amazon Web Services account** column does not appear, change your the **Discovered resources** page preferences. For more information, see Changing preferences for Discovered resource page.

   - To view other pages of resources, choose a page number, or navigate by using **<** or **>** to view next or previous pages.

   - To see the latest information about resources in the page, choose **Refresh**.

Filtering discovered resources

You can use one or more filters on the **Telemetry config** page or the **Discovered resources** page to change your view of the resources. Your filter settings persist across both pages.

To filter resources on the Telemetry config page

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.

2. In the navigation pane, choose **Settings**, and then choose **Telemetry config**.

3. You can filter the discovered resources that are displayed on the page by specifying an account ID or tag value.

   a. Choose **Find resource**.

   b. Choose **Account ID** or **Tag value**, and then choose additional options for the filter. Statistics about telemetry coverage for each resource change based on your filter options.

4. To remove a filter, in the filter text box, choose **X**.

To filter resources on the Discovered resources page

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Telemetry config**.
3. To view all resource types discovered by telemetry configuration or to view one resource type, do one of the following:

   a. To view all resources discovered by telemetry configuration, choose **View resources**. The **Discovered resources** page appears and shows all resources discovered by telemetry configuration.

   b. To view one resource type, at the bottom of the page, choose a type of Amazon resource. The **Discovered resources** page appears. The **Discovered resources** page filters all discovered resources for that resource type.

4. You can filter the resources displayed in the page based on any of the columns in the page. You can change the columns in the page by changing your preferences for the **Discovered resources** page. For more information, see Changing preferences for Discovered resource page.

   a. Choose **Find resource**. Filters for each column in the page appear. Choose one, then choose additional options to define the filter. Resources appear in the page that match the filter settings.

   b. To further filter the resources displayed in the page, choose **Find resources** again, choose another filter, and choose additional options. Resources appear in the page that match all of the filter settings.

5. To remove one of the filters, in the filter text box, choose **X**.
6. To remove all of the filters and see all resource types discovered by telemetry configuration, choose **Clear filters**.

Changing preferences for Discovered resource page

You can change your preferences for the **Discovered resources** page to control how many resources appear per page and which detailed metrics appear in the page. Only detailed metrics in view can be used to filter the resources displayed in the discovered resources page. For more information, see Filtering discovered resources.

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Telemetry config**.
3. Choose **View resources**. The **Discovered resources** page appears.
4. Choose the gear icon.
5. In the **Preferences** dialog box, choose the number of resources per page and the visible content to show as columns.
6. Choose **Confirm**.

Turning off CloudWatch telemetry auditing

When you no longer need telemetry configuration, turn it off. When telemetry configuration is turned off, it no longer shows the status of telemetry for resources in your account or organization. You can turn on telemetry configuration at any time. For more information, see Turning on CloudWatch telemetry auditing.

1. Open the CloudWatch console.
2. In the navigation pane, choose **Telemetry config**.
3. Choose **Turn off**.

Infrastructure monitoring

The topics in this section explain CloudWatch features that can help you gain operational visibility into your Amazon resources.

Topics

Container Insights

Use CloudWatch Container Insights to collect, aggregate, and summarize metrics and logs from your containerized applications and microservices. Container Insights is available for Amazon Elastic Container Service (Amazon ECS), Amazon Elastic Kubernetes Service (Amazon EKS), RedHat OpenShift on Amazon (ROSA), and Kubernetes platforms on Amazon EC2. Container Insights supports collecting metrics from clusters deployed on Amazon Fargate for both Amazon ECS and Amazon EKS.

CloudWatch automatically collects metrics for many resources, such as CPU, memory, disk, and network. Container Insights also provides diagnostic information, such as container restart failures, to help you isolate issues and resolve them quickly. You can also set CloudWatch alarms on metrics that Container Insights collects.

Container Insights collects data as *performance log events* using embedded metric format. These performance log events are entries that use a structured JSON schema that enables high-cardinality data to be ingested and stored at scale. From this data, CloudWatch creates aggregated metrics at the cluster, node, pod, task, and service level as CloudWatch metrics. The metrics that Container Insights collects are available in CloudWatch automatic dashboards, and are also viewable in the **Metrics** section of the CloudWatch console. Metrics are not visible until the container tasks have been running for some time.

When you deploy Container Insights, it automatically creates a log group for the performance log events. You don't need to create this log group yourself.

To help you manage your Container Insights costs, CloudWatch does not automatically create all possible metrics from the log data. However, you can view additional metrics and additional levels of granularity by using CloudWatch Logs Insights to analyze the raw performance log events.

With the original version of Container Insights, metrics collected and logs ingested are charged as custom metrics. With Container Insights with enhanced observability for Amazon EKS, Container Insights metrics and logs are charged per observation instead of being charged per metric stored or log ingested. For more information about CloudWatch pricing, see Amazon CloudWatch Pricing.

In Amazon EKS, RedHatOpenshift on Amazon, and Kubernetes, Container Insights uses a containerized version of the CloudWatch agent to discover all of the running containers in a cluster. It then collects performance data at every layer of the performance stack.

Container Insights supports encryption with the Amazon KMS key for the logs and metrics that it collects. To enable this encryption, you must manually enable Amazon KMS encryption for the log group that receives Container Insights data. This causes Container Insights to encrypt this data using the provided KMS key. Only symmetric keys are supported. Do not use asymmetric KMS keys to encrypt your log groups.

For more information, see Encrypt Log Data in CloudWatch Logs Using Amazon KMS.

Supported platforms

Container Insights is available for Amazon Elastic Container Service, Amazon Elastic Kubernetes Service, RedHat OpenShift on Amazon, and Kubernetes platforms on Amazon EC2 instances.

- For Amazon ECS, Container Insights collects metrics at the cluster, task, and service levels on both Linux and Windows Server instances. Container Insights collects metrics at the instance level only on Linux instances. Network metrics are available for containers that use `bridge` network mode and `awsvpc` network mode, but are not available for containers that use `host` network mode.
- For Amazon Elastic Kubernetes Service, and Kubernetes platforms on Amazon EC2 instances, Container Insights is supported on both Linux and Windows instances.

Container Insights with enhanced observability for Amazon ECS

On December 2, 2024, Amazon released Container Insights with enhanced observability for Amazon ECS. This version supports enhanced observability for Amazon ECS clusters using the

Amazon EC2 and Fargate launch types. After you configure Container Insights with enhanced observability on Amazon ECS, Container Insights auto-collects detailed infrastructure telemetry from the cluster level down to the container level in your environment and displays these critical performance data in curated dashboards removing the heavy lifting in observability set-up. For information about how to set up Container Insights with enhanced observability, see Setting up Container Insights on Amazon ECS.

Container Insights with enhanced observability provides all of the Container Insights metrics, plus additional task and container metrics. For more information, see Amazon ECS Container Insights with enhanced observability metrics.

Container Insights with enhanced observability also supports CloudWatch cross-account observability. You can use a single monitoring account to monitor and troubleshoot applications that span multiple Amazon accounts within a single Region. For more information, see CloudWatch cross-account observability.

Container Insights with enhanced observability for Amazon EKS

On November 6, 2023, a new version of Container Insights was released. This version supports enhanced observability for Amazon EKS clusters running on Amazon EC2 and can collect more detailed metrics from these clusters. After installation, it automatically collects detailed infrastructure telemetry and container logs for your Amazon EKS clusters. You can then use curated, immediately usable dashboards to drill down into application and infrastructure telemetry.

Container Insights with enhanced observability for Amazon EKS collects granular health, performance, and status metrics up to the container level, and also control plane metrics. For more information about the additional metrics and dimensions collected, see Amazon EKS and Kubernetes Container Insights with enhanced observability metrics.

If you installed Container Insights by using the CloudWatch agent on an Amazon EKS cluster on Amazon EC2 after November 6, 2023, you have Container Insights with enhanced observability for Amazon EKS. Otherwise, you can upgrade an Amazon EKS cluster to this new version by following the instructions in Upgrading to Container Insights with enhanced observability for Amazon EKS in CloudWatch.

Container Insights supports CloudWatch cross-account observability. You use a single monitoring account to monitor and troubleshoot your applications that span multiple Amazon accounts within a single Region. For more information, see CloudWatch cross-account observability.

Container Insights with enhanced observability for Amazon EKS also supports Windows worker nodes.

Container Insights with enhanced observability for Amazon EKS is not supported on Fargate.

> **Note**
>
> You can find whether you have clusters that can be upgraded to Container Insights with enhanced observability for Amazon EKS by navigating to the Container Insights console. To do so, choose **Insights**, **Container Insights** in the navigation pane of the CloudWatch console. In the Container Insights console, a banner informs you if you have any Amazon EKS clusters that can be upgraded, and links to the upgrade page.

CloudWatch agent container image

Amazon provides a CloudWatch agent container image on Amazon Elastic Container Registry. For more information, see cloudwatch-agent on Amazon ECR.

Setting up Container Insights

The Container Insights setup process is different for Amazon ECS and Amazon EKS and Kubernetes.

Topics

Setting up Container Insights on Amazon ECS

You can use one or both of the following options to enable Container Insights on Amazon ECS clusters:

- Use the Amazon Web Services Management Console or the Amazon CLI to start collecting cluster-level, task-level, and service-level metrics.
- Deploy the CloudWatch agent as a daemon service to start collecting of instance-level metrics on clusters that are hosted on Amazon EC2 instances.

Topics

Setting up Container Insights on Amazon ECS

You can set up Container Insights with enhanced observability or Container Insights on new and existing Amazon ECS clusters using either the Amazon ECS console or the Amazon CLI. Container Insights collects metrics at the cluster, task, and service levels. Container Insights with enhanced observability provides additional dimensions and metrics, allowing you to deep dive down to container level visibility.

If you're using Amazon ECS on an Amazon EC2 instance, launch that instance using an AMI that includes Amazon ECS agent version 1.29 or later. For information about updating your agent version, see Updating the Amazon ECS Container Agent.

> **Note**
>
> If the customer managed Amazon KMS key that you use for your Amazon ECS Container Insights metrics is not already configured to work with CloudWatch, you must update the key policy to allow for encrypted logs in CloudWatch Logs. You must also associate your own Amazon KMS key with the log group in `/aws/ecs/containerinsights/`*`ClusterName`*`/performance`. For more information, see Encrypt log data in CloudWatch Logs using Amazon Key Management Service.

We recommend that you use Container Insights with enhanced observability instead of Container Insights as it provides detailed visibility in your container environment, reducing the mean time to resolution.

Set up Container Insights with enhanced observability

You can turn on Container Insights with enhanced observability using the Amazon ECS console or Amazon CLI.

Amazon CLI

Use the following command to turn on Container Insights with enhanced observability.

Set the `containerInsights` account setting to `enhanced`

```
aws ecs put-account-setting --name containerInsights --value enhanced
```

Example output

```
{
    "setting": {
        "name": "containerInsights",
        "value": "enhanced",
        "principalArn": "arn:aws:iam::123456789012:johndoe",
         "type": user
    }
}
```

> **Note**
>
> By default, the `put-account-setting` applies only to the currently authenticated user. To enable the setting account-wide for all users and roles, use the root user as in the following example.
>
> ```
> aws ecs put-account-setting --name containerInsights --value enhanced --
> principal-arn arn:aws:iam::accountID:root
> ```

After you set this account setting, all new clusters automatically use Container Insights with enhanced observability. Use the `update-cluster-settings` command to add Container Insights with enhanced observability to existing cluster, or to upgrade clusters that currently use Container Insights to Container Insights with enhanced observability.

```
aws ecs update-cluster-settings --cluster cluster-name --settings
 name=containerInsights,value=enhanced
```

Amazon ECS console

1. Open the console at https://console.amazonaws.cn/ecs/v2.

2. In the navigation bar at the top, select the Region for which to view your account settings.
3. In the navigation page, choose **Account Settings**.
4. Choose **Update**.
5. To use Container Insights with enhanced observability, choose **Container Insights with enhanced observability**.
6. Choose **Save changes**.
7. On the confirmation screen, choose **Confirm** to save the selection.

After you set this, all new clusters automatically use Container Insights with enhanced observability. You can add Container Insights with enhanced observability to existing clusters, or update clusters that currently use Container Insights to Container Insights with enhanced observability. For more information, see Updating an Amazon ECS cluster in the *Amazon Elastic Container Service Developer Guide*.

Set up Container Insights

You can turn on Container Insights using the Amazon ECS console or Amazon CLI.

Amazon CLI

To use Container Insights, set the `container Insights` account setting to `enabled`. Use the following command to turn on Container Insights.

```
aws ecs put-account-setting --name containerInsights --value enabled
```

Example output

```
{
    "setting": {
        "name": "container Insights",
        "value": "enabled",
        "principalArn": "arn:aws:iam::123456789012:johndoe",
         "type": user
    }
}
```

When you set the `container Insights` account setting to `enabled`, all new clusters have Container Insights enabled by default. Use the `update-cluster-settings` command to add Container Insights to an existing cluster.

```
aws ecs update-cluster-settings --cluster cluster-name --settings
 name=containerInsights,value=enabled
```

Amazon ECS console

1. Open the console at https://console.amazonaws.cn/ecs/v2.
2. In the navigation bar at the top, select the Region for which to view your account settings.
3. In the navigation page, choose **Account Settings**.
4. Choose **Update**.
5. To use Container Insights, choose **Container Insights**.
6. Choose **Save changes**.
7. On the confirmation screen, choose **Confirm** to save the selection.

After you set this, all new clusters automatically use Container Insights. Update existing clusters to add Container Insights. For more information, see Updating an Amazon ECS cluster in the *Amazon Elastic Container Service Developer Guide*.

Setting up Container Insights on Amazon ECS using Amazon Distro for OpenTelemetry

Use this section if you want to use Amazon Distro for OpenTelemetry to set up CloudWatch Container Insights on an Amazon ECS cluster. For more information about Amazon Distro for Open Telemetry, see Amazon Distro for OpenTelemetry.

These steps assume that you already have a cluster running Amazon ECS. For more information about using Amazon Distro for Open Telemetry with Amazon ECS and setting up an Amazon ECS cluster for this purpose, see Setting up Amazon Distro for OpenTelemetry Collector in Amazon Elastic Container Service.

Step 1: Create a task role

The first step is creating a task role in the cluster that the Amazon OpenTelemetry Collector will use.

To create a task role for Amazon Distro for OpenTelemetry

1. Open the IAM console at https://console.amazonaws.cn/iam/.
2. In the navigation pane, choose **Policies** and then choose **Create policy**.
3. Choose the **JSON** tab and copy in the following policy:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents",
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:DescribeLogStreams",
                "logs:DescribeLogGroups",
                "ssm:GetParameters"
            ],
            "Resource": "*"
        }
    ]
}
```

4. Choose **Review policy**.
5. For name, enter **`AWSDistroOpenTelemetryPolicy`**, and then choose **Create policy**.
6. In the left navigation pane, choose **Roles** and then choose **Create role**.
7. In the list of services, choose **Elastic Container Service**.
8. Lower on the page, choose **Elastic Container Service Task** and then choose **Next: Permissions**.
9. In the list of policies, search for **AWSDistroOpenTelemetryPolicy**.
10. Select the check box next to **AWSDistroOpenTelemetryPolicy**.
11. Choose **Next: Tags** and then choose **Next: Review.**
12. For **Role name** enter **`AWSOpenTelemetryTaskRole`** and then choose **Create role**.

Step 2: Create a task execution role

The next step is creating a task execution role for the Amazon OpenTelemetry Collector.

To create a task execution role for Amazon Distro for OpenTelemetry

1. Open the IAM console at https://console.amazonaws.cn/iam/.
2. In the left navigation pane, choose **Roles** and then choose **Create role**.
3. In the list of services, choose **Elastic Container Service**.
4. Lower on the page, choose **Elastic Container Service Task** and then choose **Next: Permissions**.
5. In the list of policies, search for **AmazonECSTaskExecutionRolePolicy** and then select the check box next to **AmazonECSTaskExecutionRolePolicy**.
6. In the list of policies, search for **CloudWatchLogsFullAccess** and then select the check box next to **CloudWatchLogsFullAccess**.
7. In the list of policies, search for **AmazonSSMReadOnlyAccess** and then select the check box next to **AmazonSSMReadOnlyAccess**.
8. Choose **Next: Tags** and then choose **Next: Review.**
9. For **Role name** enter **`AWSOpenTelemetryTaskExecutionRole`** and then choose **Create role**.

Step 3: Create a task definition

The next step is creating a task definition.

To create a task definition for Amazon Distro for OpenTelemetry

1. Open the console at https://console.amazonaws.cn/ecs/v2.
2. In the navigation pane, choose **Task definitions**
3. Choose **Create new task definition**, **Create new task definition**.
4. For **Task definition family**, specify a unique name for the task definition.
5. Configure your containers, and then choose **Next**.
6. Under **Metrics and logging**, select **Use metric collection**.
7. Choose **Next**.
8. Choose **Create**.

For more information about using the Amazon OpenTelemetry collector with Amazon ECS, see Setting up Amazon Distro for OpenTelemetry Collector in Amazon Elastic Container Service.

Step 4: Run the task

The final step is running the task that you've created.

To run the task for Amazon Distro for OpenTelemetry

1. Open the console at https://console.amazonaws.cn/ecs/v2.
2. In the left navigation pane, choose **Task Definitions** and then select the task that you just created.
3. Choose **Actions**, **Deploy**, **Run task**.
4. Choose **Deploy**, **Run task**.
5. In the **Compute options** section, from **Existing cluster**, choose the cluster.
6. Choose **Create**.
7. Next, you can check for the new metrics in the CloudWatch console.
8. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
9. In the left navigation pane, choose **Metrics**.

   You should see a **ECS/ContainerInsights** namespace. Choose that namespace and you should see eight metrics.

Deploying the CloudWatch agent to collect EC2 instance-level metrics on Amazon ECS

To deploy the CloudWatch agent to collect instance-level metrics from Amazon ECS clusters that are hosted on EC2 instance, use a quick start setup with a default configuration, or install the agent manually to be able to customize it.

Both methods require that you already have at least one Amazon ECS cluster deployed with an EC2 launch type and that the CloudWatch agent continer has access to the Amazon EC2 Instance Metadata Service (IMDS). For more information about IMDS, see [Instance metadata and user data](#).

These methods also assume that you have the Amazon CLI installed. Additionally, to run the commands in the following procedures, you must be logged on to an account or role that has the **IAMFullAccess** and **AmazonECS_FullAccess** policies.

Topics

- [Quick setup using Amazon CloudFormation](#)

- Manual and custom setup

Quick setup using Amazon CloudFormation

To use the quick setup, enter the following command to use Amazon CloudFormation to install the agent. Replace *cluster-name* and *cluster-region* with the name and Region of your Amazon ECS cluster.

This command creates the IAM roles **CWAgentECSTaskRole** and **CWAgentECSExecutionRole**. If these roles already exist in your account, use `ParameterKey=CreateIAMRoles,ParameterValue=False` instead of `ParameterKey=CreateIAMRoles,ParameterValue=True` when you enter the command. Otherwise, the command will fail.

```
ClusterName=cluster-name
Region=cluster-region
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/ecs-task-definition-templates/deployment-mode/daemon-service/cwagent-ecs-instance-metric/cloudformation-quickstart/cwagent-ecs-instance-metric-cfn.json
aws cloudformation create-stack --stack-name CWAgentECS-${ClusterName}-${Region} \
    --template-body file://cwagent-ecs-instance-metric-cfn.json \
    --parameters ParameterKey=ClusterName,ParameterValue=${ClusterName} \
                 ParameterKey=CreateIAMRoles,ParameterValue=True \
    --capabilities CAPABILITY_NAMED_IAM \
    --region ${Region}
```

(Alternative) Using your own IAM roles

If you want to use your own custom ECS task role and ECS task execution role instead of the **CWAgentECSTaskRole** and **CWAgentECSExecutionRole** roles, first make sure that the role to be used as the ECS task role has **CloudWatchAgentServerPolicy** attached. Also, make sure that the role to be used as the ECS task execution role has both the **CloudWatchAgentServerPolicy** and **AmazonECSTaskExecutionRolePolicy** policies attached. Then enter the following command. In the command, replace *task-role-arn* with the ARN of your custom ECS task role, and replace *execution-role-arn* with the ARN of your custom ECS task execution role.

```
ClusterName=cluster-name
Region=cluster-region
TaskRoleArn=task-role-arn
ExecutionRoleArn=execution-role-arn
```

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/ecs-task-definition-templates/deployment-mode/daemon-service/cwagent-ecs-instance-metric/cloudformation-quickstart/cwagent-ecs-instance-metric-cfn.json
aws cloudformation create-stack --stack-name CWAgentECS-${ClusterName}-${Region} \
    --template-body file://cwagent-ecs-instance-metric-cfn.json \
    --parameters ParameterKey=ClusterName,ParameterValue=${ClusterName} \
                 ParameterKey=TaskRoleArn,ParameterValue=${TaskRoleArn} \
                 ParameterKey=ExecutionRoleArn,ParameterValue=${ExecutionRoleArn} \
    --capabilities CAPABILITY_NAMED_IAM \
    --region ${Region}
```

Troubleshooting the quick setup

To check the status of the Amazon CloudFormation stack, enter the following command.

```
ClusterName=cluster-name
Region=cluster-region
aws cloudformation describe-stacks --stack-name CWAgentECS-$ClusterName-$Region --region $Region
```

If you see the `StackStatus` is other than `CREATE_COMPLETE` or `CREATE_IN_PROGRESS`, check the stack events to find the error. Enter the following command.

```
ClusterName=cluster-name
Region=cluster-region
aws cloudformation describe-stack-events --stack-name CWAgentECS-$ClusterName-$Region
 --region $Region
```

To check the status of the `cwagent` daemon service, enter the following command. In the output, you should see that the `runningCount` is equal to the `desiredCount` in the `deployment` section. If it isn't equal, check the `failures` section in the output.

```
ClusterName=cluster-name
Region=cluster-region
aws ecs describe-services --services cwagent-daemon-service --cluster $ClusterName --region $Region
```

You can also use the CloudWatch Logs console to check the agent log. Look for the **/ecs/ecs-cwagent-daemon-service** log group.

Deleting the Amazon CloudFormation stack for the CloudWatch agent

If you need to delete the Amazon CloudFormation stack, enter the following command.

```
ClusterName=cluster-name
Region=cluster-region
aws cloudformation delete-stack --stack-name CWAgentECS-${ClusterName}-${Region} --
region ${Region}
```

Manual and custom setup

Follow the steps in this section to manually deploy the CloudWatch agent to collect instance-level metrics from your Amazon ECS clusters that are hosted on EC2 instances.

Necessary IAM roles and policies

Two IAM roles are required. You must create them if they don't already exist. For more information about these roles, see IAM roles for Tasks and Amazon ECS Task Execution Role.

- An *ECS task role*, which is used by the CloudWatch agent to publish metrics. If this role already exists, you must make sure it has the `CloudWatchAgentServerPolicy` policy attached.
- An *ECS task execution role*, which is used by Amazon ECS agent to launch the CloudWatch agent. If this role already exists, you must make sure it has the `AmazonECSTaskExecutionRolePolicy` and `CloudWatchAgentServerPolicy` policies attached.

If you do not already have these roles, you can use the following commands to create them and attach the necessary policies. This first command creates the ECS task role.

```
aws iam create-role --role-name CWAgentECSTaskRole \
    --assume-role-policy-document "{\"Version\": \"2012-10-17\",\"Statement\": [{\"Sid
\": \"\",\"Effect\": \"Allow\",\"Principal\": {\"Service\": \"ecs-tasks.amazonaws.com
\"},\"Action\": \"sts:AssumeRole\"}]}"
```

After you enter the previous command, note the value of `Arn` from the command output as "TaskRoleArn". You'll need to use it later when you create the task definition. Then enter the following command to attach the necessary policies.

```
aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/
CloudWatchAgentServerPolicy \
    --role-name CWAgentECSTaskRole
```

This next command creates the ECS task execution role.

```
aws iam create-role --role-name CWAgentECSExecutionRole \
    --assume-role-policy-document "{\"Version\": \"2012-10-17\",\"Statement\": [{\"Sid
\": \"\",\"Effect\": \"Allow\",\"Principal\": {\"Service\": \"ecs-tasks.amazonaws.com
\"},\"Action\": \"sts:AssumeRole\"}]}"
```

After you enter the previous command, note the value of `Arn` from the command output as "ExecutionRoleArn". You'll need to use it later when you create the task definition. Then enter the following commands to attach the necessary policies.

```
aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/
CloudWatchAgentServerPolicy \
    --role-name CWAgentECSExecutionRole

aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/service-role/
AmazonECSTaskExecutionRolePolicy \
    --role-name CWAgentECSExecutionRole
```

Create the task definition and launch the daemon service

Create a task definition and use it to launch the CloudWatch agent as a daemon service. To create the task definition, enter the following command. In the first lines, replace the placeholders with the actual values for your deployment. *`logs-region`* is the Region where CloudWatch Logs is located, and *`cluster-region`* is the Region where your cluster is located. *`task-role-arn`* is the Arn of the ECS task role that you are using, and *`execution-role-arn`* is the Arn of the ECS task execution role.

```
TaskRoleArn=task-role-arn
ExecutionRoleArn=execution-role-arn
AWSLogsRegion=logs-region
Region=cluster-region
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
insights/latest/ecs-task-definition-templates/deployment-mode/daemon-service/cwagent-
ecs-instance-metric/cwagent-ecs-instance-metric.json \
    | sed "s|{{task-role-arn}}|${TaskRoleArn}|;s|{{execution-role-arn}}|
${ExecutionRoleArn}|;s|{{awslogs-region}}|${AWSLogsRegion}|" \
    | xargs -0 aws ecs register-task-definition --region ${Region} --cli-input-json
```

Then run the following command to launch the daemon service. Replace *`cluster-name`* and *`cluster-region`* with the name and Region of your Amazon ECS cluster.

> **Important**
>
> Remove all capacity provider strategies before you run this command. Otherwise, the command won't work.

```
ClusterName=cluster-name
Region=cluster-region
aws ecs create-service \
    --cluster ${ClusterName} \
    --service-name cwagent-daemon-service \
    --task-definition ecs-cwagent-daemon-service \
    --scheduling-strategy DAEMON \
    --region ${Region}
```

If you see this error message, `An error occurred (InvalidParameterException) when calling the CreateService operation: Creation of service was not idempotent`, you have already created a daemon service named `cwagent-daemon-service`. You must delete that service first, using the following command as an example.

```
ClusterName=cluster-name
Region=cluster-region
aws ecs delete-service \
    --cluster ${ClusterName} \
    --service cwagent-daemon-service \
    --region ${Region} \
    --force
```

(Optional) Advanced configuration

Optionally, you can use SSM to specify other configuration options for the CloudWatch agent in your Amazon ECS clusters that are hosted on EC2 instances. These options are as follows:

- `metrics_collection_interval` – How often in seconds that the CloudWatch agent collects metrics. The default is 60. The range is 1–172,000.
- `endpoint_override` – (Optional) Specifies a different endpoint to send logs to. You might want to do this if you're publishing from a cluster in a VPC and you want the logs data to go to a VPC endpoint.

  The value of `endpoint_override` must be a string that is a URL.

- `force_flush_interval` – Specifies in seconds the maximum amount of time that logs remain in the memory buffer before being sent to the server. No matter the setting for this field, if the size of the logs in the buffer reaches 1 MB, the logs are immediately sent to the server. The default value is 5 seconds.
- `region` – By default, the agent publishes metrics to the same Region where the Amazon ECS container instance is located. To override this, you can specify a different Region here. For example, `"region" : "us-east-1"`

The following is an example of a customized configuration:

```
{
    "agent": {
        "region": "us-east-1"
    },
    "logs": {
        "metrics_collected": {
            "ecs": {
                "metrics_collection_interval": 30
            }
        },
        "force_flush_interval": 5
    }
}
```

To customize your CloudWatch agent configuration in your Amazon ECS containers

1. Make sure that the **AmazonSSMReadOnlyAccess** policy is attached to your Amazon ECS Task Execution role. You can enter the following command to do so. This example assumes that your Amazon ECS Task Execution role is CWAgentECSExecutionRole. If you are using a different role, substitute that role name in the following command.

```
aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/
AmazonSSMReadOnlyAccess \
        --role-name CWAgentECSExecutionRole
```

2. Create the customized configuration file similar to the preceding example. Name this file `/tmp/ecs-cwagent-daemon-config.json`.

3. Run the following command to put this configuration into the Parameter Store. Replace *cluster-region* with the Region of your Amazon ECS cluster. To run this command, you must be logged on to a user or role that has the **AmazonSSMFullAccess** policy.

```
Region=cluster-region
aws ssm put-parameter \
    --name "ecs-cwagent-daemon-service" \
    --type "String" \
    --value "`cat /tmp/ecs-cwagent-daemon-config.json`" \
    --region $Region
```

4. Download the task definition file to a local file, such as `/tmp/cwagent-ecs-instance-metric.json`

```
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/ecs-task-definition-templates/deployment-mode/daemon-service/cwagent-ecs-instance-metric/cwagent-ecs-instance-metric.json -o /tmp/cwagent-ecs-instance-metric.json
```

5. Modify the task definition file. Remove the following section:

```
"environment": [
                {
                    "name": "USE_DEFAULT_CONFIG",
                    "value": "True"
                }
            ],
```

Replace that section with the following:

```
"secrets": [
                {
                    "name": "CW_CONFIG_CONTENT",
                    "valueFrom": "ecs-cwagent-daemon-service"
                }
            ],
```

6. Restart the agent as a daemon service by following these steps:

    a. Run the following command.

```
TaskRoleArn=task-role-arn
ExecutionRoleArn=execution-role-arn
AWSLogsRegion=logs-region
Region=cluster-region
cat /tmp/cwagent-ecs-instance-metric.json \
    | sed "s|{{task-role-arn}}|${TaskRoleArn}|;s|{{execution-role-arn}}|
${ExecutionRoleArn}|;s|{{awslogs-region}}|${AWSLogsRegion}|" \
    | xargs -0 aws ecs register-task-definition --region ${Region} --cli-input-
json
```

b. Run the following command to launch the daemon service. Replace *`cluster-name`* and *`cluster-region`* with the name and Region of your Amazon ECS cluster.

```
ClusterName=cluster-name
Region=cluster-region
aws ecs create-service \
    --cluster ${ClusterName} \
    --service-name cwagent-daemon-service \
    --task-definition ecs-cwagent-daemon-service \
    --scheduling-strategy DAEMON \
    --region ${Region}
```

If you see this error message, `An error occurred (InvalidParameterException) when calling the CreateService operation: Creation of service was not idempotent`, you have already created a daemon service named `cwagent-daemon-service`. You must delete that service first, using the following command as an example.

```
ClusterName=cluster-name
Region=Region
aws ecs delete-service \
    --cluster ${ClusterName} \
    --service cwagent-daemon-service \
    --region ${Region} \
    --force
```

Deploying the Amazon Distro for OpenTelemetry to collect EC2 instance-level metrics on Amazon ECS clusters

Use the steps in this section to use Amazon Distro for OpenTelemetry to collect EC2 instance-level metrics on an Amazon ECS cluster. For more information about the Amazon Distro for OpenTelemetry, see Amazon Distro for OpenTelemetry.

These steps assume that you already have a cluster running Amazon ECS. This cluster must be deployed with the EC2 launch type. For more information about using Amazon Distro for Open Telemetry with Amazon ECS and setting up an Amazon ECS cluster for this purpose, see Setting up Amazon Distro for OpenTelemetry Collector in Amazon Elastic Container Service for ECS EC2 instance level metrics.

Topics

Quick setup using Amazon CloudFormation

Download the Amazon CloudFormation template file for installing the Amazon Distro for OpenTelemetry collector for Amazon ECS on EC2. Run the following curl command.

```
curl -O https://raw.githubusercontent.com/aws-observability/aws-otel-collector/main/
deployment-template/ecs/aws-otel-ec2-instance-metrics-daemon-deployment-cfn.yaml
```

After you download the template file, open it and replace *PATH_TO_CloudFormation_TEMPLATE* with the path where you saved the template file. Then export the following parameters and run the Amazon CloudFormation command, as shown in the following command.

- **Cluster_Name**– The Amazon ECS cluster name
- **AWS_Region**– The Region where the data will be sent
- **PATH_TO_CloudFormation_TEMPLATE**– The path where you saved the Amazon CloudFormation template file.
- **command**– To enable the Amazon Distro for OpenTelemetry collector to collect the instance-level metrics for Amazon ECS on Amazon EC2, you must specify `--config=/etc/ecs/otel-instance-metrics-config.yaml` for this parameter.

```
ClusterName=Cluster_Name
Region=AWS_Region
command=--config=/etc/ecs/otel-instance-metrics-config.yaml
aws cloudformation create-stack --stack-name AOCECS-${ClusterName}-${Region} \
--template-body file://PATH_TO_CloudFormation_TEMPLATE \
--parameters ParameterKey=ClusterName,ParameterValue=${ClusterName} \
ParameterKey=CreateIAMRoles,ParameterValue=True \
ParameterKey=command,ParameterValue=${command} \
--capabilities CAPABILITY_NAMED_IAM \
--region ${Region}
```

After running this command, use the Amazon ECS console to see if the task is running.

Troubleshooting the quick setup

To check the status of the Amazon CloudFormation stack, enter the following command.

```
ClusterName=cluster-name
Region=cluster-region
aws cloudformation describe-stack --stack-name AOCECS-$ClusterName-$Region --region
 $Region
```

If the value of `StackStatus` is anything other than `CREATE_COMPLETE` or `CREATE_IN_PROGRESS`, check the stack events to find the error. Enter the following command.

```
ClusterName=cluster-name
Region=cluster-region
aws cloudformation describe-stack-events --stack-name AOCECS-$ClusterName-$Region --
region $Region
```

To check the status of the `AOCECS` daemon service, enter the following command. In the output, you should see that `runningCount` is equal to the `desiredCount` in the deployment section. If it isn't equal, check the failures section in the output.

```
ClusterName=cluster-name
Region=cluster-region
aws ecs describe-services --services AOCECS-daemon-service --cluster $ClusterName --
region $Region
```

You can also use the CloudWatch Logs console to check the agent log. Look for the **/aws/ecs/containerinsights/{ClusterName}/performance** log group.

Manual and custom setup

Follow the steps in this section to manually deploy the Amazon Distro for OpenTelemetry to collect instance-level metrics from your Amazon ECS clusters that are hosted on Amazon EC2 instances.

Step 1: Necessary roles and policies

Two IAM roles are required. You must create them if they don't already exist. For more information about these roles, see Create IAM policy and Create IAM role.

Step 2: Create the task definition

Create a task definition and use it to launch the Amazon Distro for OpenTelemetry as a daemon service.

To use the task definition template to create the task definition, follow the instructions in Create ECS EC2 Task Definition for EC2 instance with Amazon OTel Collector.

To use the Amazon ECS console to create the task definition, follow the instructions in Install Amazon OTel Collector by creating Task Definition through Amazon console for Amazon ECS EC2 instance metrics.

Step 3: Launch the daemon service

To launch the Amazon Distro for OpenTelemetry as a daemon service, follow the instructions in Run your task on the Amazon Elastic Container Service (Amazon ECS) using daemon service.

(Optional) Advanced configuration

Optionally, you can use SSM to specify other configuration options for the Amazon Distro for OpenTelemetry in your Amazon ECS clusters that are hosted on Amazon EC2 instances. For more information, about creating a configuration file, see Custom OpenTelemetry Configuration. For more information about the options that you can use in the configuration file, see Amazon Container Insights Receiver.

Set up FireLens to send logs to CloudWatch Logs

FireLens for Amazon ECS enables you to use task definition parameters to route logs to Amazon CloudWatch Logs for log storage and analytics. FireLens works with Fluent Bit and Fluentd. We provide an Amazon for Fluent Bit image, or you can use your own Fluent Bit or Fluentd image.

Creating Amazon ECS task definitions with a FireLens configuration is supported using the Amazon SDKs, Amazon CLI, and Amazon Web Services Management Console. For more information about CloudWatch Logs, see What is CloudWatch Logs?.

There are key considerations when using FireLens for Amazon ECS. For more information, see Considerations.

To find the Amazon for Fluent Bit images, see Using the Amazon for Fluent Bit image.

To create a task definition that uses a FireLens configuration, see Creating a task definition that uses a FireLens configuration.

Example

The following task definition example demonstrates how to specify a log configuration that forwards logs to a CloudWatch Logs log group. For more information, see What Is Amazon CloudWatch Logs? in the *Amazon CloudWatch Logs User Guide*.

In the log configuration options, specify the log group name and the Region it exists in. To have Fluent Bit create the log group on your behalf, specify `"auto_create_group":"true"`. You can also specify the task ID as the log stream prefix, which assists in filtering. For more information, see Fluent Bit Plugin for CloudWatch Logs.

```
{
 "family": "firelens-example-cloudwatch",
 "taskRoleArn": "arn:aws:iam::123456789012:role/ecs_task_iam_role",
 "containerDefinitions": [
  {
   "essential": true,
   "image": "906394416424.dkr.ecr.us-west-2.amazonaws.com/aws-for-fluent-bit:latest",
   "name": "log_router",
   "firelensConfiguration": {
    "type": "fluentbit"
   },
   "logConfiguration": {
    "logDriver": "awslogs",
    "options": {
     "awslogs-group": "firelens-container",
     "awslogs-region": "us-west-2",
     "awslogs-create-group": "true",
     "awslogs-stream-prefix": "firelens"
    }
```

```
    },
    "memoryReservation": 50
    },
    {
     "essential": true,
     "image": "nginx",
     "name": "app",
     "logConfiguration": {
      "logDriver":"awsfirelens",
      "options": {
      "Name": "cloudwatch_logs",
      "region": "us-west-2",
      "log_key": "log",
                                "log_group_name": "/aws/ecs/containerinsights/my-
cluster/application",
      "auto_create_group": "true",
      "log_stream_name": "my-task-id"
     }
    },
    "memoryReservation": 100
   }
  ]
}
```

Setting up Container Insights on Amazon EKS and Kubernetes

Container Insights is supported on Amazon EKS versions 1.23 and later. The quick start method of installation is supported only on versions 1.24 and later.

The overall process for setting up Container Insights on Amazon EKS or Kubernetes is as follows:

1. Verify that you have the necessary prerequisites.
2. Set up the Amazon CloudWatch Observability EKS add-on, the CloudWatch agent, or Amazon Distro for OpenTelemetry on your cluster to send metrics to CloudWatch.

> **Note**
>
> To use Container Insights with enhanced observability for Amazon EKS, you must use the Amazon CloudWatch Observability EKS add-on or the CloudWatch agent. For more information about this version of Container Insights, see Container Insights with enhanced observability for Amazon EKS.

To use Container Insights with Fargate, you must use Amazon Distro for OpenTelemetry. Container Insights with enhanced observability for Amazon EKS is not supported on Fargate.

Note

Container Insights now supports Windows worker nodes in an Amazon EKS cluster. Container Insights with enhanced observability for Amazon EKS is also supported on Windows. For information about enabling Container Insights on Windows, see Using the CloudWatch agent with Container Insights enhanced observability enabled.

Set up Fluent Bit or Fluentd to send logs to CloudWatch Logs. (This is enabled by default if you install the Amazon CloudWatch Observability EKS add-on.)

You can perform these steps at once as part of the quick start setup if you are using the CloudWatch agent, or do them separately.

3. (Optional) Set up Amazon EKS control plane logging.
4. (Optional) Set up the CloudWatch agent as a StatsD endpoint on the cluster to send StatsD metrics to CloudWatch.
5. (Optional) Enable App Mesh Envoy Access Logs.

With the original version of Container Insights, metrics collected and logs ingested are charged as custom metrics. With Container Insights with enhanced observability for Amazon EKS, Container Insights metrics and logs are charged per observation instead of being charged per metric stored or log ingested. For more information about CloudWatch pricing, see Amazon CloudWatch Pricing.

Topics

Verifying prerequisites for Container Insights in CloudWatch

Before you install Container Insights on Amazon EKS or Kubernetes, verify the following. These prerequisites apply whether you are using the CloudWatch agent or Amazon Distro for OpenTelemetry to set up Container Insights on Amazon EKS clusters.

- You have a functional Amazon EKS or Kubernetes cluster with nodes attached in one of the Regions that supports the Container Insights for Amazon EKS and Kubernetes. For the list of supported Regions, see Container Insights.
- You have `kubectl` installed and running. For more information, see Installing `kubectl` in the *Amazon EKS User Guide*.
- If you're using Kubernetes running on Amazon instead of using Amazon EKS, the following prerequisites are also necessary:
  - Be sure that your Kubernetes cluster has enabled role-based access control (RBAC). For more information, see Using RBAC Authorization in the Kubernetes Reference.
  - Your kubelet has enabled Webhook authorization mode. For more information, see Kubelet authentication/authorization in the Kubernetes Reference.

You must also grant IAM permissions to enable your Amazon EKS worker nodes to send metrics and logs to CloudWatch. There are two ways to do this:

- Attach a policy to the IAM role of your worker nodes. This works for both Amazon EKS clusters and other Kubernetes clusters.
- Use an IAM role for service accounts for the cluster, and attach the policy to this role. This works only for Amazon EKS clusters.

The first option grants permissions to CloudWatch for the entire node, while using an IAM role for the service account gives CloudWatch access to only the appropriate daemonset pods.

Attaching a policy to the IAM role of your worker nodes

Follow these steps to attach the policy to the IAM role of your worker nodes. This works for both Amazon EKS clusters and Kubernetes clusters outside of Amazon EKS.

To attach the necessary policy to the IAM role for your worker nodes

1. Open the Amazon EC2 console at https://console.amazonaws.cn/ec2/.

2. Select one of the worker node instances and choose the IAM role in the description.
3. On the IAM role page, choose **Attach policies**.
4. In the list of policies, select the check box next to **CloudWatchAgentServerPolicy**. If necessary, use the search box to find this policy.
5. Choose **Attach policies**.

If you're running a Kubernetes cluster outside Amazon EKS, you might not already have an IAM role attached to your worker nodes. If not, you must first attach an IAM role to the instance and then add the policy as explained in the previous steps. For more information on attaching a role to an instance, see Attaching an IAM Role to an Instance in the *Amazon EC2 User Guide*.

If you're running a Kubernetes cluster outside Amazon EKS and you want to collect EBS volume IDs in the metrics, you must add another policy to the IAM role attached to the instance. Add the following as an inline policy. For more information, see Adding and Removing IAM Identity Permissions in the *IAM User Guide*.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "ec2:DescribeVolumes"
            ],
            "Resource": "*",
            "Effect": "Allow"
        }
    ]
}
```

Using an IAM service account role

This method works only on Amazon EKS clusters.

To grant permission to CloudWatch using an IAM service account role

1. If you haven't already, enable IAM roles for service accounts on your cluster. For more information, see Enabling IAM roles for service accounts on your cluster .

2. If you haven't already, configure the service account to use an IAM role. For more information, see Configuring a Kubernetes service account to assume an IAM role.

   When you create the role, attach the **CloudWatchAgentServerPolicy** IAM policy to the role in addition to the policy that you create for the role. Also, the associated Kubernetes Service Account that is linked to this role should be created in the `amazon-cloudwatch` namespace, where the CloudWatch and Fluent Bit daemonsets will be deployed in the upcoming steps
3. If you haven't already, associate the IAM role with a service account in your cluster. For more information, see Configuring a Kubernetes service account to assume an IAM role.

Using the CloudWatch agent with Container Insights enhanced observability enabled

Use the instructions in one of the following sections to set up Container Insights on an Amazon EKS cluster or Kubernetes cluster by using the CloudWatch agent. The quick start instructions are supported only on Amazon EKS versions 1.24 and later.

> **Note**
>
> You can install Container Insights by following the instructions in any one of the following sections. You don't need to follow all three sets of instructions.

Topics

- Quick start with the Amazon CloudWatch Observability EKS add-on
- Quick Start setup for Container Insights on Amazon EKS and Kubernetes
- Setting up the CloudWatch agent to collect cluster metrics

Quick start with the Amazon CloudWatch Observability EKS add-on

You can use the Amazon EKS add-on to install Container Insights with enhanced observability for Amazon EKS. The add-on installs the CloudWatch agent to send infrastructure metrics from the cluster, installs Fluent Bit to send container logs, and also enables CloudWatch Application Signals to send application performance telemetry.

When you use the Amazon EKS add-on version 1.5.0 or later, Container Insights is enabled on both Linux and Windows worker nodes in the cluster. Currently, Application Signals is not supported on Windows in Amazon EKS.

The Amazon EKS add-on is not supported for clusters running Kubernetes instead of Amazon EKS.

For more information about the Amazon CloudWatch Observability EKS add-on, see Install the CloudWatch agent with the Amazon CloudWatch Observability EKS add-on or the Helm chart.

If you use version 3.1.0 or later of the add-on, you can use EKS Pod Identity to grant the required permissions to the add-on. EKS Pod Identity is the recommended option and provides benefits such as least privilege, credential rotation, and auditability. Additionally, using EKS Pod Identity allows you to install the EKS add-on as part of the cluster creation itself.

To install the Amazon CloudWatch Observability EKS add-on

1. Follow the EKS Pod Identity association steps to create the IAM role and set up the EKS Pod Identity agent.
2. Attach an IAM policy that grants the required permissions to your role. Replace *my-role* with the name of your IAM role from the previous step.

```
aws iam attach-role-policy \
 --role-name my-role \
--policy-arn=arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy
```

3. Enter the following command, using with the IAM role you created in the previous step:

```
aws eks create-addon \
--addon-name amazon-cloudwatch-observability \
--cluster-name my-cluster-name \
--pod-identity-associations serviceAccount=cloudwatch-
agent,roleArn=arn:aws:iam::111122223333:role/my-role
```

Quick Start setup for Container Insights on Amazon EKS and Kubernetes

> **Important**
>
> If you are installing Container Insights on an Amazon EKS cluster, we recommend that you use the Amazon CloudWatch Observability EKS add-on for the installation, instead of using the instructions in this section. Additionally, to retrieve accelerated computing networks, you must use the Amazon CloudWatch Observability EKS add-on. For more information and instructions, see Quick start with the Amazon CloudWatch Observability EKS add-on.

To complete the setup of Container Insights, you can follow the quick start instructions in this section. If you are installing in an Amazon EKS cluster and you use the instructions in this section on or after November 6, 2023, you install Container Insights with enhanced observability for Amazon EKS in the cluster.

> **Important**
>
> Before completing the steps in this section, you must have verified the prerequisites including IAM permissions. For more information, see Verifying prerequisites for Container Insights in CloudWatch.

Alternatively, you can instead follow the instructions in the following two sections, Setting up the CloudWatch agent to collect cluster metrics and Send logs to CloudWatch Logs. Those sections provide more configuration details on how the CloudWatch agent works with Amazon EKS and Kubernetes, but require you to perform more installation steps.

With the original version of Container Insights, metrics collected and logs ingested are charged as custom metrics. With Container Insights with enhanced observability for Amazon EKS, Container Insights metrics and logs are charged per observation instead of being charged per metric stored or log ingested. For more information about CloudWatch pricing, see Amazon CloudWatch Pricing.

> **Note**
>
> Amazon has now launched Fluent Bit as the default log solution for Container Insights with significant performance gains. We recommend that you use Fluent Bit instead of Fluentd.

Quick Start with the CloudWatch agent operator and Fluent Bit

There are two configurations for Fluent Bit: an optimized version and a version that provides an experience more similar to Fluentd. The Quick Start configuration uses the optimized version. For more details about the Fluentd-compatible configuration, see Set up Fluent Bit as a DaemonSet to send logs to CloudWatch Logs.

The CloudWatch agent operator is an additional container that gets installed to an Amazon EKS cluster. It is modeled after the OpenTelemetry Operator for Kubernetes. The operator manages the lifecycle of Kubernetes resources in a cluster. It installs the CloudWatch Agent, DCGM Exporter

(NVIDIA), and the Amazon Neuron Monitor on an Amazon EKS cluster and manages them. Fluent Bit and the CloudWatch Agent for Windows are installed directly to an Amazon EKS cluster without the operator managing them.

For a more secure and feature-rich certificate authority solution, the CloudWatch agent operator requires cert-manager, a widely-adopted solution for TLS certificate management in Kubernetes. Using cert-manager simplifies the process of obtaining, renewing, managing and using these certificates. It ensures that certificates are valid and up to date, and attempts to renew certificates at a configured time before expiry. cert-manager also facilitates issuing certificates from a variety of supported sources, including Amazon Certificate Manager Private Certificate Authority.

To deploy Container Insights using the quick start

1. Install cert-manager if it is not already installed in the cluster. For more information, see [cert-manager Installation](https://cert-manager.io/docs/installation/).
2. Install the custom resource definitions (CRD) by entering the following commmand.

   ```
   curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/main/k8s-quickstart/cwagent-custom-resource-definitions.yaml | kubectl
    apply --server-side -f -
   ```

3. Install the operator by entering the following command. Replace *my-cluster-name* with the name of your Amazon EKS or Kubernetes cluster, and replace *my-cluster-region* with the name of the Region where the logs are published. We recommend that you use the same Region where your cluster is deployed to reduce the Amazon outbound data transfer costs.

   ```
   ClusterName=my-cluster-name
   RegionName=my-cluster-region
   curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/main/k8s-quickstart/cwagent-operator-rendered.yaml | sed 's/{{cluster_name}}/'${ClusterName}'/g;s/{{region_name}}/'${RegionName}'/g' | kubectl
    apply -f -
   ```

   For example, to deploy Container Insights on the cluster named `MyCluster` and publish the logs and metrics to US West (Oregon), enter the following command.

   ```
   ClusterName='MyCluster'
   RegionName='us-west-2'
   curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/main/k8s-quickstart/cwagent-operator-rendered.yaml | sed 's/
   ```

```
{{cluster_name}}/'${ClusterName}'/g;s/{{region_name}}/'${RegionName}'/g' | kubectl
 apply -f -
```

Migrating from Container Insights

If you already have Container Insights configured in an Amazon EKS cluster and you want to migrate to Container Insights with enhanced observability for Amazon EKS, see [Upgrading to Container Insights with enhanced observability for Amazon EKS in CloudWatch](#)

Deleting Container Insights

If you want to remove Container Insights after using the quick start setup, enter the following commands.

```
ClusterName=my-cluster-name
RegionName=my-cluster-region
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-
container-insights/main/k8s-quickstart/cwagent-operator-rendered.yaml | sed 's/
{{cluster_name}}/'${ClusterName}'/g;s/{{region_name}}/'${RegionName}'/g' | kubectl
 delete -f -
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
insights/main/k8s-quickstart/cwagent-custom-resource-definitions.yaml | kubectl delete
 -f -
```

Setting up the CloudWatch agent to collect cluster metrics

> **Important**
>
> If you are installing Container Insights on on Amazon EKS cluster, we recommend that you use the Amazon CloudWatch Observability EKS add-on for the installation, instead of using the instructions in this section. For more information and instructions, see [Quick start with the Amazon CloudWatch Observability EKS add-on](#).

To set up Container Insights to collect metrics, you can follow the steps in [Quick Start setup for Container Insights on Amazon EKS and Kubernetes](#) or you can follow the steps in this section. In the following steps, you set up the CloudWatch agent to be able to collect metrics from your clusters.

If you are installing in an Amazon EKS cluster and you use the instructions in this section on or after November 6, 2023, you install Container Insights with enhanced observability for Amazon EKS in the cluster.

Step 1: Create a namespace for CloudWatch

Use the following step to create a Kubernetes namespace called `amazon-cloudwatch` for CloudWatch. You can skip this step if you have already created this namespace.

To create a namespace for CloudWatch

- Enter the following command.

```
kubectl apply -f https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/daemonset/container-insights-monitoring/cloudwatch-namespace.yaml
```

Step 2: Create a service account in the cluster

Use one of the following methods to create a service account for the CloudWatch agent, if you do not already have one.

- Use `kubectl`
- Use a `kubeconfig` file

Use `kubectl` for authentication

To use `kubectl` to create a service account for the CloudWatch agent

- Enter the following command.

```
kubectl apply -f https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/daemonset/container-insights-monitoring/cwagent/cwagent-serviceaccount.yaml
```

If you didn't follow the previous steps, but you already have a service account for the CloudWatch agent that you want to use, you must ensure that it has the following rules. Additionally, in the rest of the steps in the Container Insights installation, you must use the name of that service account instead of `cloudwatch-agent`.

```
rules:
  - apiGroups: [""]
    resources: ["pods", "nodes", "endpoints"]
    verbs: ["list", "watch"]
  - apiGroups: [ "" ]
    resources: [ "services" ]
    verbs: [ "list", "watch" ]
  - apiGroups: ["apps"]
    resources: ["replicasets", "daemonsets", "deployments", "statefulsets"]
    verbs: ["list", "watch"]
  - apiGroups: ["batch"]
    resources: ["jobs"]
    verbs: ["list", "watch"]
  - apiGroups: [""]
    resources: ["nodes/proxy"]
    verbs: ["get"]
  - apiGroups: [""]
    resources: ["nodes/stats", "configmaps", "events"]
    verbs: ["create", "get"]
  - apiGroups: [""]
    resources: ["configmaps"]
    resourceNames: ["cwagent-clusterleader"]
    verbs: ["get","update"]
  - nonResourceURLs: ["/metrics"]
    verbs: ["get", "list", "watch"]
```

Use `kubeconfig` for authentication

Alternatively, you can use a `kubeconfig` file for authentication. This method allows you to bypass the need for a service account b directly specifying the `kubeconfig` path in your CloudWatch agent configuration. It also allows you to remove your dependency on the Kubernetes control plane API for authentication, streamlining your setup and potentially increasing security by managing authentication through your kubeconfig file.

To use this method, update your CloudWatch agent configuration file to specify the path to your `kubeconfig` file, as in the following example.

```
{
  "logs": {
```

```
    "metrics_collected": {
      "kubernetes": {
        "cluster_name": "YOUR_CLUSTER_NAME",
        "enhanced_container_insights": false,
        "accelerated_compute_metrics": false,
        "tag_service": false,
        "kube_config_path": "/path/to/your/kubeconfig"
        "host_ip": "HOSTIP"
      }
    }
  }
}
```

To create a `kubeconfig` file, create a Certificate Signing Request (CSR) for the `admin/{create_your_own_user}` user with the `system:masters` Kubernetes role. Then sign with Kubernetes cluster's Certificate Authority (CA) and create the `kubeconfig` file.

Step 3: Create a ConfigMap for the CloudWatch agent

Use the following steps to create a ConfigMap for the CloudWatch agent.

To create a ConfigMap for the CloudWatch agent

1. Download the ConfigMap YAML to your `kubectl` client host by running the following command:

   ```
   curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/daemonset/container-insights-monitoring/cwagent/cwagent-configmap-enhanced.yaml
   ```

2. Edit the downloaded YAML file, as follows:

   - **cluster_name** – In the `kubernetes` section, replace `{{cluster_name}}` with the name of your cluster. Remove the `{{}}` characters. Alternatively, if you're using an Amazon EKS cluster, you can delete the `"cluster_name"` field and value. If you do, the CloudWatch agent detects the cluster name from the Amazon EC2 tags.

3. (Optional) Make further changes to the ConfigMap based on your monitoring requirements, as follows:

- **metrics_collection_interval** – In the `kubernetes` section, you can specify how often the agent collects metrics. The default is 60 seconds. The default cadvisor collection interval in kubelet is 15 seconds, so don't set this value to less than 15 seconds.
- **endpoint_override** – In the `logs` section, you can specify the CloudWatch Logs endpoint if you want to override the default endpoint. You might want to do this if you're publishing from a cluster in a VPC and you want the data to go to a VPC endpoint.
- **force_flush_interval** – In the `logs` section, you can specify the interval for batching log events before they are published to CloudWatch Logs. The default is 5 seconds.
- **region** – By default, the agent published metrics to the Region where the worker node is located. To override this, you can add a `region` field in the `agent` section: for example, `"region":"us-west-2"`.
- **statsd** section – If you want the CloudWatch Logs agent to also run as a StatsD listener in each worker node of your cluster, you can add a `statsd` section to the `metrics` section, as in the following example. For information about other StatsD options for this section, see [Retrieve custom metrics with StatsD](#).

```
"metrics": {
  "metrics_collected": {
    "statsd": {
      "service_address":":8125"
    }
  }
}
```

A full example of the JSON section is as follows. If you're using a `kubeconfig` file for authentication, add the `kube_config_path` parameter to specify the path to your kubeconfig file.

```
{
    "agent": {
        "region": "us-east-1"
    },
    "logs": {
        "metrics_collected": {
            "kubernetes": {
                "cluster_name": "MyCluster",
                "metrics_collection_interval": 60,
```

```
                "kube_config_path": "/path/to/your/kubeconfig" //if using
 kubeconfig for authentication
            }
        },
        "force_flush_interval": 5,
        "endpoint_override": "logs.us-east-1.amazonaws.com"
    },
    "metrics": {
        "metrics_collected": {
            "statsd": {
                "service_address": ":8125"
            }
        }
    }
}
```

4. Create the ConfigMap in the cluster by running the following command.

```
kubectl apply -f cwagent-configmap-enhanced.yaml
```

Step 4: Deploy the CloudWatch agent as a DaemonSet

To finish the installation of the CloudWatch agent and begin collecting container metrics, use the following steps.

To deploy the CloudWatch agent as a DaemonSet

1. • If you do not want to use StatsD on the cluster, enter the following command.

   ```
   kubectl apply -f https://raw.githubusercontent.com/aws-samples/amazon-
   cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/
   deployment-mode/daemonset/container-insights-monitoring/cwagent/cwagent-
   daemonset.yaml
   ```

   • If you do want to use StatsD, follow these steps:

     a. Download the DaemonSet YAML to your `kubectl` client host by running the following command.

```
curl -O  https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-
container-insights/latest/k8s-deployment-manifest-templates/deployment-
mode/daemonset/container-insights-monitoring/cwagent/cwagent-daemonset.yaml
```

b. Uncomment the `port` section in the `cwagent-daemonset.yaml` file as in the following:

```
ports:
  - containerPort: 8125
    hostPort: 8125
    protocol: UDP
```

c. Deploy the CloudWatch agent in your cluster by running the following command.

```
kubectl apply -f cwagent-daemonset.yaml
```

d. Deploy the CloudWatch agent on Windows nodes in your cluster by running the following command. The StatsD listener is not supported on the CloudWatch agent on Windows.

```
kubectl apply -f cwagent-daemonset-windows.yaml
```

2. Validate that the agent is deployed by running the following command.

```
kubectl get pods -n amazon-cloudwatch
```

When complete, the CloudWatch agent creates a log group named `/aws/containerinsights/Cluster_Name/performance` and sends the performance log events to this log group. If you also set up the agent as a StatsD listener, the agent also listens for StatsD metrics on port 8125 with the IP address of the node where the application pod is scheduled.

Troubleshooting

If the agent doesn't deploy correctly, try the following:

- Run the following command to get the list of pods.

```
kubectl get pods -n amazon-cloudwatch
```

- Run the following command and check the events at the bottom of the output.

```
kubectl describe pod pod-name -n amazon-cloudwatch
```

- Run the following command to check the logs.

```
kubectl logs pod-name  -n amazon-cloudwatch
```

Using Amazon Distro for OpenTelemetry

You can set up Container Insights to collect metrics from Amazon EKS clusters by using the Amazon Distro for OpenTelemetry collector. For more information about the Amazon Distro for OpenTelemetry, see Amazon Distro for OpenTelemetry.

> **Important**
>
> If you install using Amazon Distro for OpenTelemetry, you install Container Insights but do not get Container Insights with enhanced observability for Amazon EKS. You will not collect the detailed metrics supported in Container Insights with enhanced observability for Amazon EKS.

How you set up Container Insights depends on whether the cluster is hosted on Amazon EC2 instances or on Amazon Fargate.

Amazon EKS clusters hosted on Amazon EC2

If you have not already done so, make sure that you have fulfilled the prerequisites including the necessary IAM roles. For more information, see Verifying prerequisites for Container Insights in CloudWatch.

Amazon provides a Helm chart that you can use to set up the monitoring of Amazon Elastic Kubernetes Service on Amazon EC2. This monitoring uses the Amazon Distro for OpenTelemetry(ADOT) Collector for metrics and Fluent Bit for logs. Therefore, the Helm chart is useful for customers who use Amazon EKS on Amazon EC2 and want to collect metrics and logs to send to CloudWatch Container Insights. For more information about this Helm chart, see ADOT Helm chart for EKS on EC2 metrics and logs to Amazon CloudWatch Container Insights.

Alternatively, you can also use the instructions in the rest of this section.

First, deploy the Amazon Distro for OpenTelemetry collector as a DaemonSet by entering the following command.

```
curl https://raw.githubusercontent.com/aws-observability/aws-otel-collector/main/
deployment-template/eks/otel-container-insights-infra.yaml |
kubectl apply -f -
```

To confirm that the collector is running, enter the following command.

```
kubectl get pods -l name=aws-otel-eks-ci -n aws-otel-eks
```

If the output of this command includes multiple pods in the `Running` state, the collector is running and collecting metrics from the cluster. The collector creates a log group named `aws/containerinsights/`*`cluster-name`*`/performance` and sends the performance log events to it.

For information about how to see your Container Insights metrics in CloudWatch, see Viewing Container Insights metrics.

Amazon has also provided documentation on GitHub for this scenario. If you want to customize the metrics and logs published by Container Insights, see https://aws-otel.github.io/docs/getting-started/container-insights/eks-infra.

Amazon EKS clusters hosted on Fargate

For instructions for how to configure and deploy an ADOT Collector to collect system metrics from workloads deployed to an Amazon EKS cluster on Fargate and send them to CloudWatch Container Insights, see Container Insights EKS Fargate in the Amazon Distro for OpenTelemetry documentation.

Send logs to CloudWatch Logs

To send logs from your containers to Amazon CloudWatch Logs, you can use Fluent Bit. For more information, see Fluent Bit.

> **Note**
>
> As of February 10 2025, Amazon has deprecated support for FluentD as a log forwarder to CloudWatch Logs. We recommend that you use Fluent Bit, which is a lightweight and resource-efficient alternative. Existing FluentD deployments will continue to function.

Migrate your logging pipeline to Fluent Bit to ensure continued support and optimal performance.
Container Insights previously also supported using FluentD to send logs from your containers. FluentD has been deprecated and is now not supported for Container Insights. Use Fluent Bit instead.

Topics

- Set up Fluent Bit as a DaemonSet to send logs to CloudWatch Logs
- (Optional) Set up Amazon EKS control plane logging
- (Optional) Enable the Use_Kubelet feature for large clusters

Set up Fluent Bit as a DaemonSet to send logs to CloudWatch Logs

The following sections help you deploy Fluent Bit to send logs from containers to CloudWatch Logs.

Topics

- Setting up Fluent Bit
- Multi-line log support
- (Optional) Reducing the log volume from Fluent Bit
- Troubleshooting
- Dashboard

Setting up Fluent Bit

To set up Fluent Bit to collect logs from your containers, you can follow the steps in Quick Start setup for Container Insights on Amazon EKS and Kubernetes or you can follow the steps in this section.

With either method, the IAM role that is attached to the cluster nodes must have sufficient permissions. For more information about the permissions required to run an Amazon EKS cluster, see Amazon EKS IAM Policies, Roles, and Permissions in the *Amazon EKS User Guide*.

In the following steps, you set up Fluent Bit as a daemonSet to send logs to CloudWatch Logs. When you complete this step, Fluent Bit creates the following log groups if they don't already exist.

> **Important**
>
> If you already have FluentD configured in Container Insights and the FluentD DaemonSet is not running as expected (this can happen if you use the `containerd` runtime), you must uninstall it before installing Fluent Bit to prevent Fluent Bit from processing the FluentD error log messages. Otherwise, you must uninstall FluentD immediately after you have successfully installed Fluent Bit. Uninstalling Fluentd after installing Fluent Bit ensures continuity in logging during this migration process. Only one of Fluent Bit or FluentD is needed to send logs to CloudWatch Logs.

Log group name	Log source
`/aws/containerinsights/ Cluster_Name /application`	All log files in `/var/log/containers`
`/aws/containerinsights/ Cluster_Name /host`	Logs from `/var/log/dmesg` , `/var/log/secure` , and `/var/log/messages`
`/aws/containerinsights/ Cluster_Name /dataplane`	The logs in `/var/log/journal` for `kubelet.service` , `kubeproxy.service` , and `docker.service` .

To install Fluent Bit to send logs from containers to CloudWatch Logs

1. If you don't already have a namespace called `amazon-cloudwatch`, create one by entering the following command:

```
kubectl apply -f https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/daemonset/container-insights-monitoring/cloudwatch-namespace.yaml
```

2. Run the following command to create a ConfigMap named `cluster-info` with the cluster name and the Region to send logs to. Replace *`cluster-name`* and *`cluster-region`* with your cluster's name and Region.

```
ClusterName=cluster-name
RegionName=cluster-region
```

```
FluentBitHttpPort='2020'
FluentBitReadFromHead='Off'
[[ ${FluentBitReadFromHead} = 'On' ]] && FluentBitReadFromTail='Off'||
 FluentBitReadFromTail='On'
[[ -z ${FluentBitHttpPort} ]] && FluentBitHttpServer='Off' ||
 FluentBitHttpServer='On'
kubectl create configmap fluent-bit-cluster-info \
--from-literal=cluster.name=${ClusterName} \
--from-literal=http.server=${FluentBitHttpServer} \
--from-literal=http.port=${FluentBitHttpPort} \
--from-literal=read.head=${FluentBitReadFromHead} \
--from-literal=read.tail=${FluentBitReadFromTail} \
--from-literal=logs.region=${RegionName} -n amazon-cloudwatch
```

In this command, the `FluentBitHttpServer` for monitoring plugin metrics is on by default. To turn it off, change the third line in the command to `FluentBitHttpPort=''` (empty string) in the command.

Also by default, Fluent Bit reads log files from the tail, and will capture only new logs after it is deployed. If you want the opposite, set `FluentBitReadFromHead='On'` and it will collect all logs in the file system.

3. Download and deploy the Fluent Bit daemonset to the cluster by running one of the following commands.

   - If you want the Fluent Bit optimized configuration for Linux computers, run this command.

     ```
     kubectl apply -f https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/daemonset/container-insights-monitoring/fluent-bit/fluent-bit.yaml
     ```

   - If you want the Fluent Bit optimized configuration for Windows computers, run this command.

     ```
     kubectl apply -f https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/daemonset/container-insights-monitoring/fluent-bit/fluent-bit-windows.yaml
     ```

- If you are using Linux computers and want the Fluent Bit configuration that is more similar to Fluentd, run this command.

```
kubectl apply -f https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/daemonset/container-insights-monitoring/fluent-bit/fluent-bit-compatible.yaml
```

Important

The Fluent Bit daemonset configuration by default sets the log level to INFO, which can result in higher CloudWatch Logs ingestion costs. If you want to reduce log ingestion volume and costs, you can change the log level to ERROR.
For more information about how to reduce the log volume, see (Optional) Reducing the log volume from Fluent Bit

4. Validate the deployment by entering the following command. Each node should have one pod named **fluent-bit-***.

```
kubectl get pods -n amazon-cloudwatch
```

The above steps create the following resources in the cluster:

- A service account named `Fluent-Bit` in the `amazon-cloudwatch` namespace. This service account is used to run the Fluent Bit daemonSet. For more information, see Managing Service Accounts in the Kubernetes Reference.
- A cluster role named `Fluent-Bit-role` in the `amazon-cloudwatch` namespace. This cluster role grants `get`, `list`, and `watch` permissions on pod logs to the `Fluent-Bit` service account. For more information, see API Overview in the Kubernetes Reference.
- A ConfigMap named `Fluent-Bit-config` in the `amazon-cloudwatch` namespace. This ConfigMap contains the configuration to be used by Fluent Bit. For more information, see Configure a Pod to Use a ConfigMap in the Kubernetes Tasks documentation.

If you want to verify your Fluent Bit setup, follow these steps.

Verify the Fluent Bit setup

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Log groups**.
3. Make sure that you're in the Region where you deployed Fluent Bit.
4. Check the list of log groups in the Region. You should see the following:
   - `/aws/containerinsights/Cluster_Name/application`
   - `/aws/containerinsights/Cluster_Name/host`
   - `/aws/containerinsights/Cluster_Name/dataplane`
5. Navigate to one of these log groups and check the **Last Event Time** for the log streams. If it is recent relative to when you deployed Fluent Bit, the setup is verified.

   There might be a slight delay in creating the `/dataplane` log group. This is normal as these log groups only get created when Fluent Bit starts sending logs for that log group.

Multi-line log support

For information on how to use Fluent Bit with multi-line logs, see the following sections of the Fluent Bit documentation:

- Multiline Parsing
- Multiline and Containers (v1.8)
- Multiline Core (v1.8)
- Always use multiline in the tail input

(Optional) Reducing the log volume from Fluent Bit

By default, we send Fluent Bit application logs and Kubernetes metadata to CloudWatch. If you want to reduce the volume of data being sent to CloudWatch, you can stop one or both of these data sources from being sent to CloudWatch. If you have followed the steps on this page to set up Fluent Bit, download the Kubernetes manifest YAML file from the kubectl `apply` command that you previously ran and modify it with your changes, which you can then re-apply to your cluster. Alternatively, if you are using the Amazon CloudWatch Observability EKS add-on or Helm chart, see (Optional) Additional configuration for information about managing the Fluent Bit configuration by using the add-on’s advanced config or the Helm chart.

To stop Fluent Bit application logs, remove the following section from the `Fluent Bit configuration` file.

```
[INPUT]
        Name                tail
        Tag                 application.*
        Path                /var/log/containers/fluent-bit*
        Parser              docker
        DB                  /fluent-bit/state/flb_log.db
        Mem_Buf_Limit       5MB
        Skip_Long_Lines     On
        Refresh_Interval    10
```

To remove Kubernetes metadata from being appended to log events that are sent to CloudWatch, add the following filters to the `application-log.conf` section in the Fluent Bit configuration. Replace *`<Metadata_1>`* and the similar fields with with the actual metadata identifiers.

```
application-log.conf: |
    [FILTER]
        Name                nest
        Match               application.*
        Operation           lift
        Nested_under        kubernetes
        Add_prefix          Kube.

    [FILTER]
        Name                modify
        Match               application.*
        Remove              Kube.<Metadata_1>
        Remove              Kube.<Metadata_2>
        Remove              Kube.<Metadata_3>

    [FILTER]
        Name                nest
        Match               application.*
        Operation           nest
        Wildcard            Kube.*
        Nested_under        kubernetes
        Remove_prefix       Kube.
```

Troubleshooting

If you don't see these log groups and are looking in the correct Region, check the logs for the Fluent Bit daemonSet pods to look for the error.

Run the following command and make sure that the status is `Running`.

```
kubectl get pods -n amazon-cloudwatch
```

If the logs have errors related to IAM permissions, check the IAM role that is attached to the cluster nodes. For more information about the permissions required to run an Amazon EKS cluster, see Amazon EKS IAM Policies, Roles, and Permissions in the *Amazon EKS User Guide*.

If the pod status is `CreateContainerConfigError`, get the exact error by running the following command.

```
kubectl describe pod pod_name -n amazon-cloudwatch
```

Dashboard

You can create a dashboard to monitor metrics of each running plugin. You can see data for input and output bytes and for record processing rates as well as output errors and retry/failed rates. To view these metrics, you will need to install the CloudWatch agent with Prometheus metrics collection for Amazon EKS and Kubernetes clusters. For more information about how to set up the dashboard, see Install the CloudWatch agent with Prometheus metrics collection on Amazon EKS and Kubernetes clusters.

> **Note**
>
> Before you can set up this dashboard, you must set up Container Insights for Prometheus metrics. For more information, see Container Insights Prometheus metrics monitoring.

To create a dashboard for the Fluent Bit Prometheus metrics

1. Create environment variables, replacing the values on the right in the following lines to match your deployment.

   ```
   DASHBOARD_NAME=your_cw_dashboard_name
   REGION_NAME=your_metric_region_such_as_us-west-1
   ```

```
CLUSTER_NAME=your_kubernetes_cluster_name
```

2. Create the dashboard by running the following command.

```
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-
prometheus/sample_cloudwatch_dashboards/fluent-bit/cw_dashboard_fluent_bit.json \
| sed "s/{{YOUR_AWS_REGION}}/${REGION_NAME}/g" \
| sed "s/{{YOUR_CLUSTER_NAME}}/${CLUSTER_NAME}/g" \
| xargs -0 aws cloudwatch put-dashboard --dashboard-name ${DASHBOARD_NAME} --
dashboard-body
```

(Optional) Set up Amazon EKS control plane logging

If you're using Amazon EKS, you can optionally enable Amazon EKS control plane logging, to provide audit and diagnostic logs directly from the Amazon EKS control plane to CloudWatch Logs. For more information, see Amazon EKS Control Plane Logging.

(Optional) Enable the Use_Kubelet feature for large clusters

By default, the Use_Kubelet feature is disabled in the FluentBit Kubernetes plugin. Enabling this feature can reduce traffic to the API server and mitigate the issue of the API Server being a bottleneck. We recommend that you enable this feature for large clusters.

To enable Use_Kubelet, first add the nodes and nodes/proxy permissions to the clusterRole config.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: fluent-bit-role
rules:
  - nonResourceURLs:
      - /metrics
    verbs:
      - get
  - apiGroups: [""]
    resources:
      - namespaces
      - pods
      - pods/logs
      - nodes
      - nodes/proxy
```

```
    verbs: ["get", "list", "watch"]
```

In the DaemonSet configuration, this feature needs host network access. The image version for `amazon/aws-for-fluent-bit` should 2.12.0 or later, or the fluent bit image version should be 1.7.2 or later.

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: fluent-bit
  namespace: amazon-cloudwatch
  labels:
    k8s-app: fluent-bit
    version: v1
    kubernetes.io/cluster-service: "true"
spec:
  selector:
    matchLabels:
      k8s-app: fluent-bit
  template:
    metadata:
      labels:
        k8s-app: fluent-bit
        version: v1
        kubernetes.io/cluster-service: "true"
    spec:
      containers:
      - name: fluent-bit
        image: amazon/aws-for-fluent-bit:2.19.0
        imagePullPolicy: Always
        env:
            - name: AWS_REGION
              valueFrom:
                configMapKeyRef:
                  name: fluent-bit-cluster-info
                  key: logs.region
            - name: CLUSTER_NAME
              valueFrom:
                configMapKeyRef:
                  name: fluent-bit-cluster-info
                  key: cluster.name
            - name: HTTP_SERVER
              valueFrom:
```

```
              configMapKeyRef:
                name: fluent-bit-cluster-info
                key: http.server
          - name: HTTP_PORT
            valueFrom:
              configMapKeyRef:
                name: fluent-bit-cluster-info
                key: http.port
          - name: READ_FROM_HEAD
            valueFrom:
              configMapKeyRef:
                name: fluent-bit-cluster-info
                key: read.head
          - name: READ_FROM_TAIL
            valueFrom:
              configMapKeyRef:
                name: fluent-bit-cluster-info
                key: read.tail
          - name: HOST_NAME
            valueFrom:
              fieldRef:
                fieldPath: spec.nodeName
          - name: HOSTNAME
            valueFrom:
              fieldRef:
                apiVersion: v1
                fieldPath: metadata.name
          - name: CI_VERSION
            value: "k8s/1.3.8"
      resources:
          limits:
            memory: 200Mi
          requests:
            cpu: 500m
            memory: 100Mi
      volumeMounts:
      # Please don't change below read-only permissions
      - name: fluentbitstate
        mountPath: /var/fluent-bit/state
      - name: varlog
        mountPath: /var/log
        readOnly: true
      - name: varlibdockercontainers
        mountPath: /var/lib/docker/containers
```

```
            readOnly: true
          - name: fluent-bit-config
            mountPath: /fluent-bit/etc/
          - name: runlogjournal
            mountPath: /run/log/journal
            readOnly: true
          - name: dmesg
            mountPath: /var/log/dmesg
            readOnly: true
        terminationGracePeriodSeconds: 10
        hostNetwork: true
        dnsPolicy: ClusterFirstWithHostNet
        volumes:
        - name: fluentbitstate
          hostPath:
            path: /var/fluent-bit/state
        - name: varlog
          hostPath:
            path: /var/log
        - name: varlibdockercontainers
          hostPath:
            path: /var/lib/docker/containers
        - name: fluent-bit-config
          configMap:
            name: fluent-bit-config
        - name: runlogjournal
          hostPath:
            path: /run/log/journal
        - name: dmesg
          hostPath:
            path: /var/log/dmesg
        serviceAccountName: fluent-bit
        tolerations:
        - key: node-role.kubernetes.io/master
          operator: Exists
          effect: NoSchedule
        - operator: "Exists"
          effect: "NoExecute"
        - operator: "Exists"
          effect: "NoSchedule"
```

The Kubernetes Plugin configuration should be similar to the following:

```
[FILTER]
        Name                kubernetes
        Match               application.*
        Kube_URL            https://kubernetes.default.svc:443
        Kube_Tag_Prefix     application.var.log.containers.
        Merge_Log           On
        Merge_Log_Key       log_processed
        K8S-Logging.Parser  On
        K8S-Logging.Exclude Off
        Labels              Off
        Annotations         Off
        Use_Kubelet         On
        Kubelet_Port        10250
        Buffer_Size         0
```

Updating or deleting Container Insights on Amazon EKS and Kubernetes

Use the steps in these sections to update your CloudWatch agent container image, or to remove Container Insights from an Amazon EKS or Kubernetes cluster.

Topics

- Upgrading to Container Insights with enhanced observability for Amazon EKS in CloudWatch
- Updating the CloudWatch agent container image
- Deleting the CloudWatch agent and Fluent Bit for Container Insights

Upgrading to Container Insights with enhanced observability for Amazon EKS in CloudWatch

> **Important**
>
> If you are upgrading or installing Container Insights on an Amazon EKS cluster, we recommend that you use the Amazon CloudWatch Observability EKS add-on for the installation, instead of using the instructions in this section. Additionally, to retrieve accelerated computing metrics, you must use the Amazon CloudWatch Observability EKS add-on. For more information and instructions, see Quick start with the Amazon CloudWatch Observability EKS add-on.

Container Insights with enhanced observability for Amazon EKS is the newest version of Container Insights. It collects detailed metrics from clusters running Amazon EKS and offers curated, immediately usable dashboards to drill down into application and infrastructure telemetry. For more information about this version of Container Insights, see [Container Insights with enhanced observability for Amazon EKS](#).

If you have installed the original version of Container Insights in an Amazon EKS cluster and you want to upgrade it to the newer version with enhanced observability, follow the instructions in this section.

> **Important**
>
> Before completing the steps in this section, you must have verified the prerequisites including cert-manager. For more information, see [Quick Start with the CloudWatch agent operator and Fluent Bit](#).

To upgrade an Amazon EKS cluster to Container Insights with enhanced observability for Amazon EKS

1. Install the CloudWatch agent operator by entering the following command. Replace *my-cluster-name* with the name of your Amazon EKS or Kubernetes cluster, and replace *my-cluster-region* with the name of the Region where the logs are published. We recommend that you use the same Region where your cluster is deployed to reduce the Amazon outbound data transfer costs.

   ```
   ClusterName=my-cluster-name
   RegionName=my-cluster-region
   curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/main/k8s-quickstart/cwagent-operator-rendered.yaml | sed 's/{{cluster_name}}/'${ClusterName}'/g;s/{{region_name}}/'${RegionName}'/g' | kubectl apply -f -
   ```

   If you notice a failure caused by conflicting resources, it is likely because you already have the CloudWatch agent and Fluent Bit with its associated components such as the ServiceAccount, the ClusterRole and the ClusterRoleBinding installed on the cluster. When the CloudWatch agent operator tries to install the CloudWatch agent and its associated components, if it detects any change in the contents, it by default fails the installation or update to avoid overwriting the state of the resources on the cluster. We recommend that you delete any existing CloudWatch agent with Container Insights setup that you had previously installed on the cluster, and then install the CloudWatch agent operator.

2. (Optional) To apply an existing custom Fluent Bit configuration, you must update the configmap associated with the Fluent Bit daemonset. The CloudWatch agent operator provides a default configuration for Fluent Bit, and you can override or modify the default configuration as needed. To apply a custom configuration, follow these steps.

   a. Open the existing configuration by entering the following command.

   ```
   kubectl edit cm fluent-bit-config -n amazon-cloudwatch
   ```

   b. Make your changes in the file, then enter `:wq` to save the file and exit edit mode.

   c. Restart Fluent Bit by entering the following command.

   ```
   kubectl rollout restart ds fluent-bit -n amazon-cloudwatch
   ```

Updating the CloudWatch agent container image

> **Important**
>
> If you are upgrading or installing Container Insights on an Amazon EKS cluster, we recommend that you use the Amazon CloudWatch Observability EKS add-on for the installation, instead of using the instructions in this section. Additionally, to retrieve accelerated computing metrics, you must use the Amazon CloudWatch Observability EKS add-on or the CloudWatch agent operator. For more information and instructions, see [Quick start with the Amazon CloudWatch Observability EKS add-on](#).

If you need to update your container image to the latest version, use the steps in this section.

To update your container image

1. Verify if the `amazoncloudwatchagent` Customer Resource Definition (CRD) already exists by entering the following command.

   ```
   kubectl get crds amazoncloudwatchagents.cloudwatch.aws.amazon.com -n amazon-
   cloudwatch
   ```

   If this command returns an error that the CRD is missing, the cluster doesn't have Container Insights with enhanced observabilit for Amazon EKS configured with the CloudWatch agent operator. In this case, see Upgrading to Container Insights with enhanced observability for Amazon EKS in CloudWatch.

2. Apply the latest `cwagent-version.yaml` file by entering the following command.

```
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
insights/main/k8s-quickstart/cwagent-version.yaml | kubectl apply -f -
```

Deleting the CloudWatch agent and Fluent Bit for Container Insights

If you installed Container Insights by using installing the CloudWatch Observability add-on for Amazon EKS, you can delete Container Insights and the CloudWatch agent by entering the following command:

> **Note**
>
> The Amazon EKS add-on now supports Container Insights on Windows worker nodes. If you delete the Amazon EKS add-on, Container Insights for Windows is also deleted.

```
aws eks delete-addon —cluster-name my-cluster —addon-name amazon-cloudwatch-
observability
```

Otherwise, to delete all resources related to the CloudWatch agent and Fluent Bit, enter the following command. In this command, *My_Cluster_Name* is the name of your Amazon EKS or Kubernetes cluster, and *My_Region* is the name of the Region where the logs are published.

```
ClusterName=My_Cluster_Name
RegionName=My-Region
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-
container-insights/main/k8s-quickstart/cwagent-operator-rendered.yaml | sed 's/
{{cluster_name}}/'${ClusterName}'/g;s/{{region_name}}/'${RegionName}'/g' | kubectl
 delete -f -
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
insights/main/k8s-quickstart/cwagent-custom-resource-definitions.yaml | kubectl delete
 -f -
```

Setting up Container Insights on RedHat OpenShift on Amazon (ROSA)

CloudWatch Container Insights with enhanced observability supports RedHat OpenShift on Amazon (ROSA). This version supports enhanced observability for ROSA clusters. After you install the CloudWatch agent operator Helm chart, Container Insights auto-collects detailed infrastructure telemetry from the cluster level down to the container level in your environment. It then displays this performance data in curated dashboards removing the heavy lifting in observability setup.

> **Note**
>
> For RedHat for OpenShift on Amazon (ROSA), when you install the CloudWatch agent operator using helm charts, the CloudWatch agent is by default also enabled to receive both metrics and traces from your applications that are instrumented for Application Signals. If you would like to optionally pass in custom configuration rules, you can do so by passing in a custom agent configuration by using the Helm chart, as outlined in (Optional) [Additional configuration], as outlined in (Optional) Additional configuration.

To install Container Insights with enhanced observability on a RedHat OpenShift on Amazon (ROSA) cluster

1. If necessary, install Helm. For more information, see Quickstart Guide in the Helm documentation.
2. Install the CloudWatch agent operator by entering the following commands. Replace *my-cluster-name* with the name of your cluster, and replace *my-cluster-region* with the Region that the cluster runs in.

```
helm repo add aws-observability https://aws-observability.github.io/helm-charts
helm repo update aws-observability
helm install --wait --create-namespace \
    --namespace amazon-cloudwatch amazon-cloudwatch-observability \
    aws-observability/amazon-cloudwatch-observability \
    --set clusterName=my-cluster-name \
    --set region=my-cluster-region \
    --set k8sMode=ROSA
```

3. Set up authorization for the agent operator by following the steps in Option 1, Option 2, or Option 3 in Install the CloudWatch agent with the Amazon CloudWatch Observability EKS add-on or the Helm chart.

Viewing Container Insights metrics

After you have Container Insights set up and it is collecting metrics, you can view those metrics in the CloudWatch console.

For Container Insights metrics to appear on your dashboard, you must complete the Container Insights setup. For more information, see Setting up Container Insights.

This procedure explains how to view the metrics that Container Insights automatically generates from the collected log data. The rest of this section explains how to further dive into your data and use CloudWatch Logs Insights to see more metrics at more levels of granularity.

To view Container Insights metrics

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Insights**, **Container Insights**.
3. Use the drop-down boxes near the top to select the type of resource to view, as well as the specific resource.

You can set a CloudWatch alarm on any metric that Container Insights collects. For more information, see Using Amazon CloudWatch alarms

> **Note**
>
> If you have already set up CloudWatch Application Insights to monitor your containerized applications, the Application Insights dashboard appears below the Container Insights dashboard. If you have not already enabled Application Insights, you can do so by choosing **Auto-configure Application Insights** below the performance view in the Container Insights dashboard.
> For more information about Application Insights and containerized applications, see Enable Application Insights for Amazon ECS and Amazon EKS resource monitoring.

Viewing the top contributors

For some of the views in Container Insights performance monitoring, you can also see the top contributors by memory or CPU, or the most recently active resources. This is available when you select any of the following dashboards in the drop-down box near the top of the page:

- ECS Services
- ECS Tasks
- EKS Namespaces
- EKS Services
- EKS Pods

When you are viewing one of these types of resources, the bottom of the page displays a table sorted initially by CPU usage. You can change it to sort by memory usage or recent activity. To see more about one of the rows in the table, you can select the checkbox next to that row and then choose **Actions** and choose one of the options in the **Actions** menu.

Using CloudWatch Logs Insights to view Container Insights data

Container Insights collects metrics by using performance log events with using embedded metric format. The logs are stored in CloudWatch Logs. CloudWatch generates several metrics automatically from the logs which you can view in the CloudWatch console. You can also do a deeper analysis of the performance data that is collected by using CloudWatch Logs Insights queries.

For more information about CloudWatch Logs Insights, see Analyze Log Data with CloudWatch Logs Insights. For more information about the log fields you can use in queries, see Container Insights performance log events for Amazon EKS and Kubernetes.

To use CloudWatch Logs Insights to query your container metric data

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Logs**, **Logs Insights**.

   Near the top of the screen is the query editor. When you first open CloudWatch Logs Insights, this box contains a default query that returns the 20 most recent log events.
3. In the box above the query editor, select one of the Container Insights log groups to query. For the following example queries to work, the log group name must end with **performance**.

   When you select a log group, CloudWatch Logs Insights automatically detects fields in the data in the log group and displays them in **Discovered fields** in the right pane. It also displays a bar graph of log events in this log group over time. This bar graph shows the distribution of events in the log group that matches your query and time range, not only the events displayed in the table.

4. In the query editor, replace the default query with the following query and choose **Run query**.

```
STATS avg(node_cpu_utilization) as avg_node_cpu_utilization by NodeName
| SORT avg_node_cpu_utilization DESC
```

This query shows a list of nodes, sorted by average node CPU utilization.

5. To try another example, replace that query with another query and choose **Run query**. More sample queries are listed later on this page.

```
STATS avg(number_of_container_restarts) as avg_number_of_container_restarts by
 PodName
| SORT avg_number_of_container_restarts DESC
```

This query displays a list of your pods, sorted by average number of container restarts.

6. If you want to try another query, you can use include fields in the list at the right of the screen. For more information about query syntax, see CloudWatch Logs Insights Query Syntax.

To see lists of your resources

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Resources**.
3. The default view is a list of your resources being monitored by Container Insights, and alarms that you have set on these resources. To see a visual map of the resources, choose **Map view**.
4. From the map view, you can pause your pointer over any resource in the map to see basic metrics about that resource. You can choose any resource to see more detailed graphs about the resource.

Use case: Seeing task-level metrics in Amazon ECS containers

The following example illustrates how to use CloudWatch Logs Insights to dive deeper into your Container Insights logs. For more examples, see the blog Introducing Amazon CloudWatch Container Insights for Amazon ECS.

Container Insights does not automatically generate metrics at the Task level of granularity. The following query displays task-level metrics for CPU and memory usage.

```
stats avg(CpuUtilized) as CPU, avg(MemoryUtilized) as Mem by TaskId, ContainerName
```

```
| sort Mem, CPU desc
```

Other sample queries for Container Insights

List of your pods, sorted by average number of container restarts

```
STATS avg(number_of_container_restarts) as avg_number_of_container_restarts by PodName
| SORT avg_number_of_container_restarts DESC
```

Pods requested vs. pods running

```
fields @timestamp, @message
| sort @timestamp desc
| filter Type="Pod"
| stats min(pod_number_of_containers) as requested,
 min(pod_number_of_running_containers) as running, ceil(avg(pod_number_of_containers-
pod_number_of_running_containers)) as pods_missing by kubernetes.pod_name
| sort pods_missing desc
```

Count of cluster node failures

```
stats avg(cluster_failed_node_count) as CountOfNodeFailures
| filter Type="Cluster"
| sort @timestamp desc
```

Application log errors by container name

```
stats count() as countoferrors by kubernetes.container_name
| filter stream="stderr"
| sort countoferrors desc
```

Metrics collected by Container Insights

Container Insights collects one set of metrics for Amazon ECS and Amazon Fargate on Amazon ECS, and a different set for Amazon EKS, Amazon Fargate on Amazon EKS, RedHat OpenShift on Amazon (ROSA), and Kubernetes.

Metrics are not visible until the container tasks have been running for some time.

Topics

- Amazon ECS Container Insights with enhanced observability metrics
- Amazon ECS Container Insights metrics
- Amazon EKS and Kubernetes Container Insights with enhanced observability metrics
- Amazon EKS and Kubernetes Container Insights metrics

Amazon ECS Container Insights with enhanced observability metrics

The following table lists the metrics and dimensions that Container Insights with enhanced observability collects for Amazon ECS. These metrics are in the `ECS/ContainerInsights` namespace. For more information, see Metrics.

If you do not see any Container Insights metrics in your console, be sure that you have completed the setup of Container Insights with enhanced observability. Metrics do not appear before Container Insights with enhanced observability has been set up completely. For more information, see Set up Container Insights with enhanced observability.

The following metrics are available for all launch types.

Metric name	Dimensions	Description
`ContainerInstanceCount`	`ClusterName`	The number of EC2 instances running the Amazon ECS agent that are registered with a cluster. This metric is collected only for container instances that are running Amazon ECS tasks in the cluster. It is not collected for empty container instances that do not have any Amazon ECS tasks. Unit: Count

Metric name	Dimensions	Description

`ContainerCpuUtilized`	`ClusterName`
	
	`ContainerName` , `TaskId`, `ServiceName` , `ClusterName`
	
	`ContainerName` , `TaskDefinitionFamily` , `ClusterName` , `TaskId`
	
	`TaskDefinitionFamily` , `ClusterName` , `ContainerName`
	
	`ServiceName` , `ClusterName` , `ContainerName`
		The CPU units used by containers in the resource that is specified by the dimension set that you're using.
		
		Unit: None

`ContainerCpuReserved`	`ClusterName`
	
	`ContainerName` , `TaskId`, `ServiceName` , `ClusterName`
	
	`ContainerName` , `TaskDefinitionFamily` , `ClusterName` , `TaskId`
	
	`TaskDefinitionFamily` , `ClusterName` , `ContainerName`
	
	`ServiceName` , `ClusterName` , `ContainerName`
		The CPU units reserved by containers in the resource that is specified by the dimension set that you're using. This metric is collected based on the CPU reservation defined in the task definition, for example, at the task or all containers level. If this is not specified in the task definition, then the instance CPU reservation is used.
		
		Unit: None

Metric name	Dimensions	Description
ContainerCpuUtilization	TaskDefinitionFamily , ClusterName ServiceName , ClusterName ClusterName ClusterName , ServiceName , TaskId ClusterName , TaskDefinitionFamily , TaskId	The total percentage of CPU units being used by containers in the resource that is specified by the dimension set that you're using. Unit: Percent
ContainerMemoryUtilized	ClusterName ContainerName , TaskId, ServiceName , ClusterName ContainerName , TaskDefinitionFamily , ClusterName , TaskId TaskDefinitionFamily , ClusterName , ContainerName ServiceName , ClusterName , ContainerName	The memory being used by containers in the resource that is specified by the dimension set that you're using. Unit: Megabytes

Metric name	Dimensions	Description

`ContainerMemoryReserved`	`ClusterName`
	
	`ContainerName` , `TaskId`, `ServiceName` , `ClusterName`
	
	`ContainerName` , `TaskDefinitionFamily` , `ClusterName` , `TaskId`
	
	`TaskDefinitionFamily` , `ClusterName` , `ContainerName`
	
	`ServiceName` , `ClusterName` , `ContainerName`
		The memory that is reserved by containers in the resource that is specified by the dimension set that you're using.
		
		This metric is collected based on the memory reservation defined in the task definition, for example, at the task or all containers level. If this is not specified in the task definition, then the instance memory reservation is used.
		
		Unit: Megabytes

Metric name	Dimensions	Description

`ContainerMemoryUtilization`	`ClusterName`
	
	`ContainerName` , `TaskId`, `ServiceName` , `ClusterName`
	
	`ContainerName` , `TaskDefinitionFamily` , `ClusterName` , `TaskId`
	
	`TaskDefinitionFamily` , `ClusterName` , `ContainerName`
	
	`ServiceName` , `ClusterName` , `ContainerName`
		The total percentage of memory being used by containers in the resource that is specified by the dimension set that you're using.
		
		Unit: Percent

`ContainerNetworkRxBytes`	`TaskDefinitionFamily` , `ClusterName`
	
	`ServiceName` , `ClusterName`
	
	`ClusterName`
	
	`ClusterName` , `ServiceName` , `TaskId`
	
	`ClusterName` , `TaskDefinitionFamily` , `TaskId`
		The number of bytes received by the container that is specified by the dimensions that you're using. This metric is obtained from the Docker runtime.
		
		This metric is available only for containers in tasks using the `awsvpc` or `bridge` network modes.
		
		Unit: Bytes/Second

Metric name	Dimensions	Description

`ContainerNetworkTxBytes`	`TaskDefinitionFamily` , `ClusterName`
	
	`ServiceName` , `ClusterName`
	
	`ClusterName`
	
	`ClusterName` , `ServiceName` , `TaskId`
	
	`ClusterName` , `TaskDefinitionFamily` , `TaskId`
		The number of bytes transmitted by the container that is specified by the dimensions that you're using. This metric is obtained from the Docker runtime.
		
		This metric is available only for containers in tasks using the `awsvpc` or `bridge` network modes.
		
		Unit: Bytes/Second

`ContainerStorageReadBytes`	`TaskDefinitionFamily` , `ClusterName`
	
	`ServiceName` , `ClusterName`
	
	`ClusterName`
	
	`ClusterName` , `ServiceName` , `TaskId`
	
	`ClusterName` , `TaskDefinitionFamily` , `TaskId`
		The number of bytes read from storage on the container in the resource that is specified by the dimensions that you're using. This does not include read bytes for your storage devices. This metric is obtained from the Docker runtime.
		
		Unit: Bytes

Metric name	Dimensions	Description

`ContainerStorageWriteBytes`	`TaskDefinitionFamily` , `ClusterName`
	
	`ServiceName` , `ClusterName`
	
	`ClusterName`
	
	`ClusterName` , `ServiceName` , `TaskId`
	
	`ClusterName` , `TaskDefinitionFamily` , `TaskId`
		The number of bytes written to storage in the container that is specified by the dimensions that you're using. This metric is obtained from the Docker runtime.
		
		Unit: Bytes

`CpuUtilized`	`TaskDefinitionFamily` , `ClusterName`
	
	`ServiceName` , `ClusterName`
	
	`ClusterName`
	
	`ClusterName` , `ServiceName` , `TaskId`
	
	`ClusterName` , `TaskDefinitionFamily` , `TaskId`
		The CPU units used by tasks in the resource that is specified by the dimension set that you're using.
		
		Unit: None

Metric name	Dimensions	Description

`CpuReserved`	`TaskDefinitionFamily` , `ClusterName`
	
	`ServiceName` , `ClusterName`
	
	`ClusterName`
	
	`ClusterName` , `ServiceName` , `TaskId`
	
	`ClusterName` , `TaskDefinitionFamily` , `TaskId`
		The CPU units reserved by tasks in the resource that is specified by the dimension set that you're using. This metric is collected based on the CPU reservation defined in the task definition, for example, at the task or all containers level. If this is not specified in the task definition, then the instance CPU reservation is used.
		
		Unit: None

`DeploymentCount`	`ServiceName` , `ClusterName`	The number of deployments in an Amazon ECS service. Unit: Count
`DesiredTaskCount`	`ServiceName` , `ClusterName`	The desired number of tasks for an Amazon ECS service. Unit: Count

Metric name	Dimensions	Description

`EBSFilesystemSize`	`ClusterName ,TaskDefinitionFamily , VolumeName`
	
	`TaskDefinitionFamily ,ClusterName`
	
	`ServiceName , ClusterName`
		The total amount, in gigabytes (GB), of Amazon EBS filesystem storage that is allocated to the resources specified by the dimensions you're using.
		
		This metric is only available for tasks that run on Amazon ECS infrastructure running on Fargate using platform version `1.4.0` or Amazon EC2 instances using container agent version `1.79.0` or later.
		
		Unit: Gigabytes (GB)

Metric name	Dimensions	Description

`EBSFilesystemUtilized`	`ClusterName` ,`TaskDefinitionFamily` , `VolumeName`
	
	`TaskDefinitionFamily` ,`ClusterName`
	
	`ServiceName` , `ClusterName`
		The total amount, in gigabytes (GB), of Amazon EBS filesystem storage that is being used by the resources specified by the dimensions that you're using.
		
		This metric is only available for tasks that run on Amazon ECS infrastructure running on Fargate using platform version `1.4.0` or Amazon EC2 instances using container agent version `1.79.0` or later.
		
		For tasks run on Fargate, Fargate reserves space on the disk that is only used by Fargate. There is no cost associated with the space Fargate uses, but you will see this additional storage using tools like `df`.
		
		Unit: Gigabytes (GB)

Metric name	Dimensions	Description

`EphemeralStorageReserved` [1]	`TaskDefinitionFamily` , `ClusterName`
	
	`ServiceName` , `ClusterName`
	
	`ClusterName`
	
	`ClusterName` , `ServiceName` , `TaskId`
	
	`ClusterName` , `TaskDefinitionFamily` , `TaskId`
		The number of bytes reserved from ephemeral storage in the resource that is specified by the dimensions that you're using. Ephemeral storage is used for the container root filesystem and any bind mount host volumes defined in the container image and task definition. The amount of ephemeral storage can’t be changed in a running task.
		
		This metric is only available for tasks that run on Fargate Linux platform version 1.4.0 or later.
		
		Unit: Gigabytes (GB)

Metric name	Dimensions	Description
MemoryUtilized	TaskDefinitionFamily ,ClusterName ServiceName , ClusterName ClusterName ClusterName , ServiceName ,TaskId ClusterName , TaskDefinitionFamily ,TaskId	The memory being used by tasks in the resource that is specified by the dimension set that you're using. Unit: Megabytes
MemoryReserved	TaskDefinitionFamily ,ClusterName ServiceName , ClusterName ClusterName ClusterName , ServiceName ,TaskId ClusterName , TaskDefinitionFamily ,TaskId	The memory that is reserved by tasks in the resource that is specified by the dimension set that you're using. This metric is collected based on the memory reservation defined in the task definition, for example, at the task or all containers level. If this is not specified in the task definition, then the instance memory reservation is used. Unit: Megabytes

Metric name	Dimensions	Description

`NetworkRxBytes`	`TaskDefinitionFamily`, `ClusterName`
	
	`ServiceName`, `ClusterName`
	
	`ClusterName`
	
	`ClusterName`, `ServiceName`, `TaskId`
	
	`ClusterName`, `TaskDefinitionFamily`, `TaskId`
		The number of bytes received by the resource that is specified by the dimensions that you're using. This metric is obtained from the Docker runtime.
		
		This metric is available only for containers in tasks using the `awsvpc` or `bridge` network modes.
		
		Unit: Bytes/Second

`NetworkTxBytes`	`TaskDefinitionFamily`, `ClusterName`
	
	`ServiceName`, `ClusterName`
	
	`ClusterName`
	
	`ClusterName`, `ServiceName`, `TaskId`
	
	`ClusterName`, `TaskDefinitionFamily`, `TaskId`
		The number of bytes transmitted by the resource that is specified by the dimensions that you're using. This metric is obtained from the Docker runtime.
		
		This metric is available only for containers in tasks using the `awsvpc` or `bridge` network modes.
		
		Unit: Bytes/Second

Metric name	Dimensions	Description
PendingTaskCount	ServiceName , ClusterName	The number of tasks currently in the PENDING state. Unit: Count
RunningTaskCount	ServiceName , ClusterName	The number of tasks currently in the RUNNING state. Unit: Count
RestartCount	ClusterName ClusterName , ServiceName ClusterName , TaskDefinitionFamily ClusterName , ServiceName , TaskId ClusterName , TaskDefinitionFamily , TaskId	The number of times a container in an Amazon ECS task has been restarted. This metric is collected only for containers that have a restart policy enabled. Unit: Count
ServiceCount	ClusterName	The number of services in the cluster. Unit: Count

Metric name	Dimensions	Description

`StorageReadBytes`	`TaskDefinitionFamily` , `ClusterName`
	
	`ServiceName` , `ClusterName`
	
	`ClusterName`
	
	`ClusterName` , `ServiceName` , `TaskId`
	
	`ClusterName` , `TaskDefinitionFamily` , `TaskId`
		The number of bytes read from storage on the instance in the resource that is specified by the dimensions that you're using. This does not include read bytes for your storage devices. This metric is obtained from the Docker runtime.
		
		Unit: Bytes

`StorageWriteBytes`	`TaskDefinitionFamily` , `ClusterName`
	
	`ServiceName` , `ClusterName`
	
	`ClusterName`
	
	`ClusterName` , `ServiceName` , `TaskId`
	
	`ClusterName` , `TaskDefinitionFamily` , `TaskId`
		The number of bytes written to storage in the resource that is specified by the dimensions that you're using. This metric is obtained from the Docker runtime.
		
		Unit: Bytes

`TaskCount`	`ClusterName`	The number of tasks running in the cluster. Unit: Count

Metric name	Dimensions	Description
TaskCpuUtilization	TaskDefinitionFamily , ClusterName ServiceName , ClusterName ClusterName ClusterName , ServiceName , TaskId ClusterName , TaskDefinitionFamily , TaskId	The total percentage of CPU units being used by a task. Unit: Percent
TaskEphemeralStorageUtilization	TaskDefinitionFamily , ClusterName ServiceName , ClusterName ClusterName ClusterName , ServiceName , TaskId ClusterName , TaskDefinitionFamily , TaskId	The total percentage of ephemeral storage being used by a task. Unit: Percent

Metric name	Dimensions	Description
TaskMemoryUtilization	TaskDefinitionFamily , ClusterName ServiceName , ClusterName ClusterName ClusterName , ServiceName , TaskId ClusterName , TaskDefinitionFamily , TaskId	The total percentage of memory being used by a task. Unit: Percent
TaskSetCount	ServiceName , ClusterName	The number of task sets in the service. Unit: Count

> **Note**
>
> The EphemeralStorageReserved and EphemeralStorageUtilized metrics are only available for tasks that run on Fargate Linux platform version 1.4.0 or later.
> Fargate reserves space on disk. It is only used by Fargate. You aren't billed for it. It isn't shown in these metrics. However, you can see this additional storage in other tools such as df.

The following metrics are available when you complete the steps in Deploying the CloudWatch agent to collect EC2 instance-level metrics on Amazon ECS and use the EC2 launch type.

Metric name	Dimensions	Description
instance_cpu_limit	ClusterName	The maximum number of CPU

Metric name	Dimensions	Description
		units that can be assigned to a single EC2 instance in the cluster. Unit: None

`instance_cpu_reserved_capacity`	`ClusterName`
	
	`InstanceId` , `ContainerInstanceId` , `ClusterName`
		The percentage of CPU currently being reserved on a single EC2 instance in the cluster.
		
		Unit: Percent

`instance_cpu_usage_total`	`ClusterName`	The number of CPU units being used on a Single EC2 instance in the cluster. Unit: None

`instance_cpu_utilization`	`ClusterName`
	
	`InstanceId` , `ContainerInstanceId` , `ClusterName`
		The total percentage of CPU units being used on a single EC2 instance in the cluster.
		
		Unit: Percent

Metric name	Dimensions	Description
instance_filesystem_utilization	ClusterName InstanceId , ContainerInstanceId , ClusterName	The total percentage of file system capacity being used on a single EC2 instance in the cluster. Unit: Percent
instance_memory_limit	ClusterName	The maximum amount of memory, in bytes, that can be assigned to a single EC2 instance in this cluster. Unit: Bytes
instance_memory_reserved_capacity	ClusterName InstanceId , ContainerInstanceId , ClusterName	The percentage of Memory currently being reserved on a single EC2 instance in the cluster. Unit: Percent

Metric name	Dimensions	Description
instance_memory_utilization	ClusterName InstanceId , ContainerInstanceId , ClusterName	The total percentage of memory being used on a single EC2 instance in the cluster. **Note** If you're using the Java ZGC garbage collector for your application, this metric might be inaccurate. Unit: Percent

Metric name	Dimensions	Description
instance_memory_working_set	ClusterName	The amount of memory, in bytes, being used on a single EC2 instance in the cluster. **Note** If you're using the Java ZGC garbage collector for your application, this metric might be inaccurate. Unit: Bytes
instance_network_total_bytes	ClusterName	The total number of bytes per second transmitted and received over the network on a single EC2 instance in the cluster. Unit: Bytes/second
instance_number_of_running_tasks	ClusterName	The number of running tasks on a single EC2 instance in the cluster. Unit: Count

Amazon ECS Container Insights metrics

The following table lists the metrics and dimensions that Container Insights collects for Amazon ECS. These metrics are in the `ECS/ContainerInsights` namespace. For more information, see Metrics.

If you do not see any Container Insights metrics in your console, be sure that you have completed the setup of Container Insights. Metrics do not appear before Container Insights has been set up completely. For more information, see Setting up Container Insights.

The following metrics are available when you complete the steps in Setting up Container Insights on Amazon ECS.

Metric name	Dimensions	Description
`ContainerInstanceCount`	`ClusterName`	The number of EC2 instances running the Amazon ECS agent that are registered with a cluster. This metric is collected only for container instances that are running Amazon ECS tasks in the cluster. It is not collected for empty container instances that do not have any Amazon ECS tasks. Unit: Count

`CpuUtilized`	`TaskDefinitionFamily` , `ClusterName`
	
	`ServiceName` , `ClusterName`
		The CPU units used by tasks in the resource that is specified by the dimension set that you're using.

Metric name	Dimensions	Description
	ClusterName	Unit: None
CpuReserved	TaskDefinitionFamily , ClusterName ServiceName , ClusterName ClusterName	The CPU units reserved by tasks in the resource that is specified by the dimension set that you're using. This metric is collected based on the CPU reservation defined in the task definition, for example, at the task or all containers level. If this is not specified in the task definition, then the instance CPU reservation is used. Unit: None
DeploymentCount	ServiceName , ClusterName	The number of deployments in an Amazon ECS service. Unit: Count
DesiredTaskCount	ServiceName , ClusterName	The desired number of tasks for an Amazon ECS service. Unit: Count

Metric name	Dimensions	Description

`EBSFilesystemSize`	`VolumeName` , `TaskDefinitionFamily` , `ClusterName`
	
	`TaskDefinitionFamily` , `ClusterName`
	
	`ServiceName` , `ClusterName`
		The total amount, in gigabytes (GB), of Amazon EBS filesystem storage that is allocated to the resources specified by the dimensions you're using.
		
		This metric is only available for tasks that run on Amazon ECS infrastructure running on Fargate using platform version `1.4.0` or Amazon EC2 instances using container agent version `1.79.0` or later.
		
		Unit: Gigabytes (GB)

Metric name	Dimensions	Description

`EBSFilesystemUtilized`	`VolumeName` , `TaskDefinitionFamily` , `ClusterName`
	
	`TaskDefinitionFamily` , `ClusterName`
	
	`ServiceName` , `ClusterName`
		The total amount, in gigabytes (GB), of Amazon EBS filesystem storage that is being used by the resources specified by the dimensions that you're using.
		
		This metric is only available for tasks that run on Amazon ECS infrastructure running on Fargate using platform version `1.4.0` or Amazon EC2 instances using container agent version `1.79.0` or later.
		
		For tasks run on Fargate, Fargate reserves space on the disk that is only used by Fargate. There is no cost associated with the space Fargate uses, but you will see this additional storage using tools like `df`.
		
		Unit: Gigabytes (GB)

Metric name	Dimensions	Description

`EphemeralStorageReserved` [1]	`TaskDefinitionFamily ,ClusterName`
	
	`ServiceName , ClusterName`
	
	`ClusterName`
		The number of bytes reserved from ephemeral storage in the resource that is specified by the dimensions that you're using. Ephemeral storage is used for the container root filesystem and any bind mount host volumes defined in the container image and task definition. The amount of ephemeral storage can't be changed in a running task.
		
		This metric is only available for tasks that run on Fargate Linux platform version 1.4.0 or later.
		
		Unit: Gigabytes (GB)

Metric name	Dimensions	Description

`EphemeralStorageUtilized` [1]	`TaskDefinitionFamily` , `ClusterName`
	
	`ServiceName` , `ClusterName`
	
	`ClusterName`
		The number of bytes used from ephemeral storage in the resource that is specified by the dimensions that you're using. Ephemeral storage is used for the container root filesystem and any bind mount host volumes defined in the container image and task definition. The amount of ephemeral storage can't be changed in a running task.
		
		This metric is only available for tasks that run on Fargate Linux platform version 1.4.0 or later.
		
		Unit: Gigabytes (GB)

Metric name	Dimensions	Description
MemoryUtilized	TaskDefinitionFamily , ClusterName ServiceName , ClusterName ClusterName	The memory being used by tasks in the resource that is specified by the dimension set that you're using. **Note** If you're using the Java ZGC garbage collector for your application, this metric might be inaccurate. Unit: Megabytes

Metric name	Dimensions	Description
MemoryReserved	TaskDefinitionFamily ,ClusterName ServiceName , ClusterName ClusterName	The memory that is reserved by tasks in the resource that is specified by the dimension set that you're using. This metric is collected based on the memory reservation defined in the task definition, for example, at the task or all containers level. If this is not specified in the task definition, then the instance memory reservation is used. Unit: Megabytes
NetworkRxBytes	TaskDefinitionFamily ,ClusterName ServiceName , ClusterName ClusterName	The number of bytes received by the resource that is specified by the dimensions that you're using. This metric is obtained from the Docker runtime. This metric is available only for containers in tasks using the awsvpc or bridge network modes. Unit: Bytes/Second

Metric name	Dimensions	Description

`NetworkTxBytes`	`TaskDefinitionFamily ,ClusterName`
	
	`ServiceName , ClusterName`
	
	`ClusterName`
		The number of bytes transmitted by the resource that is specified by the dimensions that you're using. This metric is obtained from the Docker runtime.
		
		This metric is available only for containers in tasks using the `awsvpc` or `bridge` network modes.
		
		Unit: Bytes/Second

`PendingTaskCount`	`ServiceName , ClusterName`	The number of tasks currently in the `PENDING` state. Unit: Count
`RunningTaskCount`	`ServiceName , ClusterName`	The number of tasks currently in the `RUNNING` state. Unit: Count

Metric name	Dimensions	Description
RestartCount	ClusterName ClusterName , ServiceName ClusterName , TaskDefinitionFamily	The number of times a container in an Amazon ECS task has been restarted. This metric is collected only for containers that have a restart policy enabled. Unit: Count
ServiceCount	ClusterName	The number of services in the cluster. Unit: Count
StorageReadBytes	TaskDefinitionFamily , ClusterName ServiceName , ClusterName ClusterName	The number of bytes read from storage on the instance in the resource that is specified by the dimensions that you're using. This does not include read bytes for your storage devices. This metric is obtained from the Docker runtime. Unit: Bytes

Metric name	Dimensions	Description

`StorageWriteBytes`	`TaskDefinitionFamily , ClusterName`
	
	`ServiceName , ClusterName`
	
	`ClusterName`
		The number of bytes written to storage in the resource that is specified by the dimensions that you're using. This metric is obtained from the Docker runtime.
		
		Unit: Bytes

`TaskCount`	`ClusterName`	The number of tasks running in the cluster. Unit: Count
`TaskSetCount`	`ServiceName , ClusterName`	The number of task sets in the service. Unit: Count

> **Note**
>
> The `EphemeralStorageReserved` and `EphemeralStorageUtilized` metrics are only available for tasks that run on Fargate Linux platform version 1.4.0 or later.
> Fargate reserves space on disk. It is only used by Fargate. You aren't billed for it. It isn't shown in these metrics. However, you can see this additional storage in other tools such as `df`.

The following metrics are available when you complete the steps in [Deploying the CloudWatch agent to collect EC2 instance-level metrics on Amazon ECS](#)

Metric name	Dimensions	Description
`instance_cpu_limit`	`ClusterName`	The maximum number of CPU

Metric name	Dimensions	Description
		units that can be assigned to a single EC2 Instance in the cluster. Unit: None

`instance_cpu_reserved_capacity`	`ClusterName`
	
	`InstanceId` , `ContainerInstanceId` , `ClusterName`
		The percentage of CPU currently being reserved on a single EC2 instance in the cluster.
		
		Unit: Percent

`instance_cpu_usage_total`	`ClusterName`	The number of CPU units being used on a Single EC2 instance in the cluster. Unit: None

`instance_cpu_utilization`	`ClusterName`
	
	`InstanceId` , `ContainerInstanceId` , `ClusterName`
		The total percentage of CPU units being used on a single EC2 instance in the cluster.
		
		Unit: Percent

Metric name	Dimensions	Description
instance_filesystem_utilization	ClusterName InstanceId , ContainerInstanceId , ClusterName	The total percentage of file system capacity being used on a single EC2 instance in the cluster. Unit: Percent
instance_memory_limit	ClusterName	The maximum amount of memory, in bytes, that can be assigned to a single EC2 Instance in this cluster. Unit: Bytes
instance_memory_reserved_capacity	ClusterName InstanceId , ContainerInstanceId , ClusterName	The percentage of Memory currently being reserved on a single EC2 Instance in the cluster. Unit: Percent

Metric name	Dimensions	Description

`instance_memory_utilization`	`ClusterName`
	
	`InstanceId` , `ContainerInstanceId` , `ClusterName`
		The total percentage of memory being used on a single EC2 Instance in the cluster.
		
		Note
		If you're using the Java ZGC garbage collector for your application, this metric might be inaccurate.
		
		Unit: Percent

Metric name	Dimensions	Description
instance_memory_working_set	ClusterName	The amount of memory, in bytes, being used on a single EC2 Instance in the cluster. **Note** If you're using the Java ZGC garbage collector for your application, this metric might be inaccurate. Unit: Bytes
instance_network_total_bytes	ClusterName	The total number of bytes per second transmitted and received over the network on a single EC2 Instance in the cluster. Unit: Bytes/second
instance_number_of_running_tasks	ClusterName	The number of running tasks on a single EC2 Instance in the cluster. Unit: Count

Amazon EKS and Kubernetes Container Insights with enhanced observability metrics

The following tables list the metrics and dimensions that Container Insights with enhanced observability collects for Amazon EKS and Kubernetes. These metrics are in the `ContainerInsights` namespace. For more information, see Metrics.

If you do not see any Container Insights with enhanced observability metrics in your console, be sure that you have completed the setup of Container Insights with enhanced observability. Metrics do not appear before Container Insights with enhanced observability has been set up completely. For more information, see Setting up Container Insights.

If you are using version 1.5.0 or later of the Amazon EKS add-on or version 1.300035.0 of the CloudWatch agent, most metrics listed in the following table are collected for both Linux and Windows nodes. See the **Metric Name** column of the table to see which metrics are not collected for Windows.

With the earlier version of Container Insights which delivers aggregated metrics at Cluster and Service level, the metrics are charged as custom metrics. With Container Insights with enhanced observability for Amazon EKS, Container Insights metrics are charged per observation instead of being charged per metric stored or log ingested. For more information about CloudWatch pricing, see Amazon CloudWatch Pricing.

> **Note**
>
> On Windows, network metrics such as `pod_network_rx_bytes` and `pod_network_tx_bytes` are not collected for host process containers.
> On RedHat OpenShift on Amazon (ROSA) clusters , diskio metrics such as `node_diskio_io_serviced_total` and `node_diskio_io_service_bytes_total` are not collected.

Metric name	Dimensions	Description
`cluster_failed_node_count`	`ClusterName`	The number of failed worker nodes in the cluster. A node is considered failed if it is suffering from any *node*

Metric name	Dimensions	Description
		conditions. For more information, see [Conditions](#) in the Kubernetes documentation.
`cluster_node_count`	`ClusterName`	The total number of worker nodes in the cluster.
`namespace_number_of_running_pods`	`Namespace ClusterName` `ClusterName`	The number of pods running per namespace in the resource that is specified by the dimensions that you're using.
`node_cpu_limit`	`ClusterName` `ClusterName ,InstanceId ,NodeName`	The maximum number of CPU units that can be assigned to a single node in this cluster.

Metric name	Dimensions	Description
node_cpu_reserved_capacity	NodeName, ClusterName , InstanceId ClusterName	The percentage of CPU units that are reserved for node components, such as kubelet, kube-proxy, and Docker. Formula: node_cpu_request / node_cpu_limit **Note** node_cpu_request is not reported directly as a metric, but is a field in performance log events. For more information, see Relevant fields in performance log events for Amazon EKS and Kubernetes.
node_cpu_usage_total	ClusterName ClusterName , InstanceId , NodeName	The number of CPU units being used on the nodes in the cluster.
node_cpu_utilization	NodeName, ClusterName , InstanceId ClusterName	The total percentage of CPU units being used on the nodes in the cluster. Formula: node_cpu_usage_total / node_cpu_limit

Metric name	Dimensions	Description

`node_filesystem_utilization`	`NodeName`, `ClusterName` , `InstanceId`
	
	`ClusterName`
		The total percentage of file system capacity being used on nodes in the cluster.
		
		Formula: `node_filesystem_usage / node_filesystem_capacity`
		
		Note
		`node_filesystem_usage` and `node_filesystem_capacity` are not reported directly as metrics, but are fields in performance log events. For more information, see [Relevant fields in performance log events for Amazon EKS and Kubernetes]().

`node_memory_limit`	`ClusterName`
	
	`ClusterName` , `InstanceId` , `NodeName`
		The maximum amount of memory, in bytes, that can be assigned to a single node in this cluster.

Metric name	Dimensions	Description

`node_filesystem_inodes`
 	
It is not available on Windows.
	`ClusterName`
	
	`ClusterName` , `InstanceId` , `NodeName`
		The total number of inodes (used and unused) on a node.

`node_filesystem_inodes_free`
 	
It is not available on Windows.
	`ClusterName`
	
	`ClusterName` , `InstanceId` , `NodeName`
		The number of unused inodes on a node.

`node_gpu_limit`	`ClusterName`
	
	`ClusterName` , `InstanceId` , `NodeName`
		The total number of GPU(s) available on the node.

`node_gpu_usage_total`	`ClusterName`
	
	`ClusterName` , `InstanceId` , `NodeName`
		The number of GPU(s) being used by the running pods on the node.

`node_gpu_reserved_capacity`	`ClusterName`
	
	`ClusterName` , `InstanceId` , `NodeName`	

Metric name	Dimensions	Description
node_memory_reserved_capacity	NodeName, ClusterName , InstanceId ClusterName	The percentage of memory currently being used on the nodes in the cluster. Formula: node_memory_request / node_memory_limit **Note** node_memory_request is not reported directly as a metric, but is a field in performance log events. For more information, see Relevant fields in performance log events for Amazon EKS and Kubernetes.
node_memory_utilization	NodeName, ClusterName , InstanceId ClusterName	The percentage of memory currently being used by the node or nodes. It is the percentage of node memory usage divided by the node memory limitation. Formula: node_memory_working_set / node_memory_limit .

Metric name	Dimensions	Description
node_memory_working_set	ClusterName ClusterName , InstanceId , NodeName	The amount of memory, in bytes, being used in the working set of the nodes in the cluster.
node_network_total_bytes	NodeName, ClusterName , InstanceId ClusterName	The total number of bytes per second transmitted and received over the network per node in a cluster. Formula: node_network_rx_bytes + node_network_tx_bytes **Note** node_network_rx_bytes and node_network_tx_bytes are not reported directly as metrics, but are fields in performance log events. For more information, see [Relevant fields in performance log events for Amazon EKS and Kubernetes]().
node_number_of_running_containers	NodeName, ClusterName , InstanceId ClusterName	The number of running containers per node in a cluster.

Metric name	Dimensions	Description

`node_number_of_running_pods`	`NodeName`, `ClusterName` , `InstanceId`
	
	`ClusterName`
		The number of running pods per node in a cluster.

`node_status_allocatable_pods`	`ClusterName`
	
	`ClusterName` , `InstanceId` , `NodeName`
		The number of pods that can be assigned to a node based on its allocatable resources, which is defined as the remainder of a node's capacity after accounting for system daemons reservations and hard eviction thresholds.

`node_status_capacity_pods`	`ClusterName`
	
	`ClusterName` , `InstanceId` , `NodeName`
		The number of pods that can be assigned to a node based on its capacity.

`node_status_condition_ready`	`ClusterName`
	
	`ClusterName` , `InstanceId` , `NodeName`
		Indicates whether the node status condition `Ready` is true for Amazon EC2 nodes.

`node_status_condition_memory_pressure`	`ClusterName`
	
	`ClusterName` , `InstanceId` , `NodeName`
		Indicates whether the node status condition `MemoryPressure` is true.

`node_status_condition_pid_pressure`	`ClusterName`
	
	`ClusterName` , `InstanceId` , `NodeName`
		Indicates whether the node status condition `PIDPressure` is true.

`node_status_condition_disk_pressure`	`ClusterName`
	
	`ClusterName` , `InstanceId` , `NodeName`
		Indicates whether the node status condition `OutOfDisk` is true.

Metric name	Dimensions	Description
node_status_condit ion_unknown	ClusterName ClusterName , InstanceI d , NodeName	Indicates whether any of the node status conditions are Unknown.
node_interface_net work_rx_dropped	ClusterName ClusterName , InstanceI d , NodeName	The number of packets which were received and subsequen tly dropped by a network interface on the node.
node_interface_net work_tx_dropped	ClusterName ClusterName , InstanceI d , NodeName	The number of packets which were due to be transmitt ed but were dropped by a network interface on the node.
node_diskio_io_ser vice_bytes_total It is not available on Windows or on ROSA clusters.	ClusterName ClusterName , InstanceI d , NodeName	The total number of bytes transferred by all I/O operations on the node.
node_diskio_io_ser viced_total It is not available on Windows or on ROSA clusters.	ClusterName ClusterName , InstanceI d , NodeName	The total number of I/O operations on the node.

Metric name	Dimensions	Description
pod_cpu_reserved_capacity	PodName, Namespace , ClusterName ClusterName ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , Service	The CPU capacity that is reserved per pod in a cluster. Formula: pod_cpu_request / node_cpu_limit **Note** pod_cpu_request is not reported directly as a metric, but is a field in performance log events. For more information, see [Relevant fields in performance log events for Amazon EKS and Kubernetes]().
pod_cpu_utilization	PodName, Namespace , ClusterName Namespace , ClusterName Service, Namespace , ClusterName ClusterName ClusterName , Namespace , PodName, FullPodName	The percentage of CPU units being used by pods. Formula: pod_cpu_usage_total / node_cpu_limit

Metric name	Dimensions	Description
pod_cpu_utilization_over_pod_limit	PodName, Namespace , ClusterName Namespace , ClusterName Service, Namespace , ClusterName ClusterName ClusterName , Namespace , PodName, FullPodName	The percentage of CPU units being used by pods relative to the pod limit. Formula: pod_cpu_usage_total / pod_cpu_limit
pod_memory_reserved_capacity	PodName, Namespace , ClusterName ClusterName ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , Service	The percentage of memory that is reserved for pods. Formula: pod_memory_request / node_memory_limit **Note** pod_memory_request is not reported directly as a metric, but is a field in performance log events. For more information, see Relevant fields in performance log events for Amazon EKS and Kubernetes.

Metric name	Dimensions	Description
pod_memory_utilization	PodName, Namespace , ClusterName Namespace , ClusterName Service, Namespace , ClusterName ClusterName ClusterName , Namespace , PodName, FullPodName	The percentage of memory currently being used by the pod or pods. Formula: pod_memory_working_set / node_memory_limit
pod_memory_utilization_over_pod_limit	PodName, Namespace , ClusterName Namespace , ClusterName Service, Namespace , ClusterName ClusterName ClusterName , Namespace , PodName, FullPodName	The percentage of memory that is being used by pods relative to the pod limit. If any containers in the pod don't have a memory limit defined, this metric doesn't appear. Formula: pod_memory_working_set / pod_memory_limit

Metric name	Dimensions	Description

`pod_network_rx_bytes`	`PodName`, `Namespace` , `ClusterName`
	
	`Namespace` , `ClusterName`
	
	`Service`, `Namespace` , `ClusterName`
	
	`ClusterName`
	
	`ClusterName` , `Namespace` , `PodName`, `FullPodName`
		The number of bytes per second being received over the network by the pod.
		
		Formula: `sum(pod_interface_network_rx_bytes)`
		
		Note
		
		`pod_interface_network_rx_bytes` is not reported directly as a metric, but is a field in performance log events. For more information, see Relevant fields in performance log events for Amazon EKS and Kubernetes.

Metric name	Dimensions	Description
pod_network_tx_bytes	PodName, Namespace , ClusterName Namespace, ClusterName Service, Namespace , ClusterName ClusterName ClusterName , Namespace , PodName, FullPodName	The number of bytes per second being transmitted over the network by the pod. Formula: sum(pod_interface_network_tx_bytes) **Note** pod_interface_network_tx_bytes is not reported directly as a metric, but is a field in performance log events. For more information, see Relevant fields in performance log events for Amazon EKS and Kubernetes.

Metric name	Dimensions	Description

`pod_cpu_request`	`ClusterName`
	
	`PodName`, `Namespace` , `ClusterName`
	
	`Namespace` , `ClusterName` , `Service`
	
	`ClusterName` , `Namespace` , `PodName`, `FullPodName`
		The CPU requests for the pod.
		
		Formula: `sum(container_cpu_request)`
		
		Note
		
		`pod_cpu_request` is not reported directly as a metric, but is a field in performance log events. For more information, see [Relevant fields in performance log events for Amazon EKS and Kubernetes]().

Metric name	Dimensions	Description

`pod_memory_request`	`ClusterName`
	
	`PodName`, `Namespace` , `ClusterName`
	
	`Namespace` , `ClusterName` , `Service`
	
	`ClusterName` , `Namespace` , `PodName`, `FullPodName`
		The memory requests for the pod.
		
		Formula: `sum(container_memory_request)`
		
		Note
		
		`pod_memory_request` is not reported directly as a metric, but is a field in performance log events. For more information, see Relevant fields in performance log events for Amazon EKS and Kubernetes.

Metric name	Dimensions	Description

`pod_cpu_limit`	`ClusterName`
	
	`PodName`, `Namespace` , `ClusterName`
	
	`Namespace` , `ClusterName` , `Service`
	
	`ClusterName` , `Namespace` , `PodName`, `FullPodName`
		The CPU limit defined for the containers in the pod. If any containers in the pod don't have a CPU limit defined, this metric doesn't appear.
		
		Formula: `sum(container_cpu_limit)`
		
		Note
		
		`pod_cpu_limit` is not reported directly as a metric, but is a field in performance log events. For more information, see Relevant fields in performance log events for Amazon EKS and Kubernetes.

Metric name	Dimensions	Description
pod_memory_limit	ClusterName PodName, Namespace , ClusterName Namespace , ClusterName , Service ClusterName , Namespace , PodName, FullPodName	The memory limit defined for the containers in the pod. If any containers in the pod don't have a memory limit defined, this metric doesn't appear. Formula: sum(container_memory_limit) **Note** pod_cpu_limit is not reported directly as a metric, but is a field in performance log events. For more information, see Relevant fields in performance log events for Amazon EKS and Kubernetes.
pod_status_failed	ClusterName PodName, Namespace , ClusterName Namespace , ClusterName , Service ClusterName , Namespace , PodName, FullPodName	Indicates that all containers in the pod have terminated, and at least one container has terminated with a non-zero status or was terminated by the system.

Metric name	Dimensions	Description

`pod_status_ready`	`ClusterName`
	
	`PodName`, `Namespace` , `ClusterName`
	
	`Namespace` , `ClusterName` , `Service`
	
	`ClusterName` , `Namespace` , `PodName`, `FullPodName`
		Indicates that all containers in the pod are ready, having reached the condition of `ContainerReady` .

`pod_status_running`	`ClusterName`
	
	`PodName`, `Namespace` , `ClusterName`
	
	`Namespace` , `ClusterName` , `Service`
	
	`ClusterName` , `Namespace` , `PodName`, `FullPodName`
		Indicates that all containers in the pod are running.

`pod_status_scheduled`	`ClusterName`
	
	`PodName`, `Namespace` , `ClusterName`
	
	`Namespace` , `ClusterName` , `Service`
	
	`ClusterName` , `Namespace` , `PodName`, `FullPodName`
		Indicates that the pod has been scheduled to a node.

Metric name	Dimensions	Description
pod_status_unknown	ClusterName PodName, Namespace , ClusterName Namespace , ClusterNa me , Service ClusterName , Namespace , PodName, FullPodName	Indicates that status of the pod can't be obtained.
pod_status_pending	ClusterName PodName, Namespace , ClusterName Namespace , ClusterNa me , Service ClusterName , Namespace , PodName, FullPodName	Indicates that the pod has been accepted by the cluster but one or more of the containers has not become ready yet.
pod_status_succeeded	ClusterName PodName, Namespace , ClusterName Namespace , ClusterNa me , Service ClusterName , Namespace , PodName, FullPodName	Indicates that all containers in the pod have successfully terminated and will not be restarted.

Metric name	Dimensions	Description
pod_number_of_containers	ClusterName PodName, Namespace , ClusterName Namespace , ClusterName , Service ClusterName , Namespace , PodName, FullPodName	Reports the number of containers defined in the pod specification.
pod_number_of_running_containers	ClusterName PodName, Namespace , ClusterName Namespace , ClusterName , Service ClusterName , Namespace , PodName, FullPodName	Reports the number of containers in the pod which are currently in the Running state.
pod_container_status_terminated	ClusterName PodName, Namespace , ClusterName Namespace , ClusterName , Service ClusterName , Namespace , PodName, FullPodName	Reports the number of containers in the pod which are in the Terminated state.

Metric name	Dimensions	Description
pod_container_status_running	ClusterName PodName, Namespace , ClusterName Namespace , ClusterName , Service ClusterName , Namespace , PodName, FullPodName	Reports the number of containers in the pod which are in the Running state.
pod_container_status_waiting	ClusterName PodName, Namespace , ClusterName Namespace , ClusterName , Service ClusterName , Namespace , PodName, FullPodName	Reports the number of containers in the pod which are in the Waiting state.
pod_container_status_waiting_reason_crash_loop_back_off	ClusterName PodName, Namespace , ClusterName Namespace , ClusterName , Service ClusterName , Namespace , PodName, FullPodName	Reports the number of containers in the pod which are pending because of a CrashLoopBackOff error, where a container repeatedly fails to start.

Metric name	Dimensions	Description
pod_container_status_waiting_reason_create_container_config_error	ClusterName PodName, Namespace , ClusterName Namespace , ClusterName , Service ClusterName , Namespace , PodName, FullPodName	Reports the number of containers in the pod which are pending with the reason CreateContainerConfigError . This is because of an error while creating the container configuration.
pod_container_status_waiting_reason_create_container_error	ClusterName PodName, Namespace , ClusterName Namespace , ClusterName , Service ClusterName , Namespace , PodName, FullPodName	Reports the number of containers in the pod which are pending with the reason CreateContainerError because of an error while creating the container.
pod_container_status_waiting_reason_image_pull_error	ClusterName PodName, Namespace , ClusterName Namespace , ClusterName , Service ClusterName , Namespace , PodName, FullPodName	Reports the number of containers in the pod which are pending because of ErrImagePull , ImagePullBackOff , or InvalidImageName . These situations are because of an error while pulling the container image.

Metric name	Dimensions	Description
pod_container_status_waiting_reason_oom_killed	ClusterName PodName, Namespace , ClusterName Namespace , ClusterName , Service ClusterName , Namespace , PodName, FullPodName	Reports the number of containers in the pod which are in the Terminated state because of running out of memory (OOM killed).
pod_container_status_waiting_reason_start_error	ClusterName PodName, Namespace , ClusterName Namespace , ClusterName , Service ClusterName , Namespace , PodName, FullPodName	Reports the number of containers in the pod which are pending with the reason being StartError because of an error while starting the container.
pod_container_status_terminated_reason_oom_killed	ContainerName , FullPodName , PodName, Namespace , ClusterName ContainerName , PodName, Namespace , ClusterName ClusterName	Indicates a pod was terminated for exceeding the memory limit. This metric is only displayed when this issue occurs.

Metric name	Dimensions	Description

`pod_interface_network_rx_dropped`	`ClusterName`
	
	`PodName`, `Namespace` , `ClusterName`
	
	`Namespace` , `ClusterName` , `Service`
	
	`ClusterName` , `Namespace` , `PodName`, `FullPodName`
		The number of packets which were received and subsequently dropped a network interface for the pod.

`pod_interface_network_tx_dropped`	`ClusterName`
	
	`PodName`, `Namespace` , `ClusterName`
	
	`Namespace` , `ClusterName` , `Service`
	
	`ClusterName` , `Namespace` , `PodName`, `FullPodName`
		The number of packets which were due to be transmitted but were dropped for the pod.

`pod_memory_working_set`	`ClusterName`
	
	`ClusterName` , `Namespace` , `PodName`
	
	`ClusterName` , `Namespace` , `Service`
	
	`ClusterName` , `Namespace` , `PodName`, `FullPodName`
		The memory in bytes that is currently being used by a pod.

Metric name	Dimensions	Description

`pod_cpu_usage_total`	`ClusterName`
	
	`ClusterName` , `Namespace` , `PodName`
	
	`ClusterName` , `Namespace` , `Service`
	
	`ClusterName` , `Namespace` , `PodName`, `FullPodName`
		The number of CPU units used by a pod.

`container_cpu_utilization`	`ClusterName`
	
	`PodName`, `Namespace` , `ClusterName` , `ContainerName`
	
	`PodName`, `Namespace` , `ClusterName` , `ContainerName` , `FullPodName`
		The percentage of CPU units being used by the container.
		
		Formula: `container_cpu_usage_total / node_cpu_limit`
		
		Note
		
		`container_cpu_utilization` is not reported directly as a metric, but is a field in performance log events. For more information, see Relevant fields in performance log events for Amazon EKS and Kubernetes.

Metric name	Dimensions	Description

`container_cpu_utilization_over_container_limit`	`ClusterName`
	
	`PodName`, `Namespace` , `ClusterName` , `ContainerName`
	
	`PodName`, `Namespace` , `ClusterName` , `ContainerName` , `FullPodName`
		The percentage of CPU units being used by the container relative to the container limit. If the container doesn't have a CPU limit defined, this metric doesn't appear.
		
		Formula: `container_cpu_usage_total / container_cpu_limit`
		
		Note
		`container_cpu_utilization_over_container_limit` is not reported directly as a metric, but is a field in performance log events. For more information, see [Relevant fields in performance log events for Amazon EKS and Kubernetes]().

Metric name	Dimensions	Description

`container_memory_utilization`	`ClusterName`
	
	`PodName`, `Namespace`, `ClusterName`, `ContainerName`
	
	`PodName`, `Namespace`, `ClusterName`, `ContainerName`, `FullPodName`
		The percentage of memory units being used by the container.
		
		Formula: `container_memory_working_set / node_memory_limit`
		
		Note
		`container_memory_utilization` is not reported directly as a metric, but is a field in performance log events. For more information, see [Relevant fields in performance log events for Amazon EKS and Kubernetes]().

Metric name	Dimensions	Description

`container_memory_utilization_over_container_limit`	`ClusterName`
	
	`PodName`, `Namespace` , `ClusterName` , `ContainerName`
	
	`PodName`, `Namespace` , `ClusterName` , `ContainerName` , `FullPodName`
		The percentage of memory units being used by the container relative to the container limit. If the container doesn't have a memory limit defined, this metric doesn't appear.
		
		Formula: `container_memory_working_set / container_memory_limit`
		
		Note
		`container_memory_utilization_over_container_limit` is not reported directly as a metric, but is a field in performance log events. For more information, see Relevant fields in performance log events for Amazon EKS and Kubernetes.

Metric name	Dimensions	Description
`container_memory_failures_total` It is not available on Windows.	`ClusterName` PodName, `Namespace` , `ClusterName` , `ContainerName` PodName, `Namespace` , `ClusterName` , `ContainerName` , `FullPodName`	The number of memory allocation failures experienced by the container.
`pod_number_of_container_restarts`	PodName, `Namespace` , `ClusterName`	The total number of container restarts in a pod.
`service_number_of_running_pods`	Service, `Namespace` , `ClusterName` `ClusterName`	The number of pods running the service or services in the cluster.
`replicas_desired`	`ClusterName` PodName, `Namespace` , `ClusterName`	The number of pods desired for a workload as defined in the workload specification.
`replicas_ready`	`ClusterName` PodName, `Namespace` , `ClusterName`	The number of pods for a workload that have reached the ready status.
`status_replicas_available`	`ClusterName` PodName, `Namespace` , `ClusterName`	The number of pods for a workload which are available . A pod is available when it has been ready for the `minReadySeconds` defined in the workload specification.

Metric name	Dimensions	Description
status_replicas_un available	ClusterName PodName, Namespace , ClusterName	The number of pods for a workload which are unavailab le. A pod is available when it has been ready for the minReadySeconds defined in the workload specification. Pods are unavailable if they have not met this criterion.
apiserver_storage_ objects	ClusterName ClusterName , resource	The number of objects stored in etcd at the time of the last check.
apiserver_storage_ db_total_size_in_b ytes	ClusterName ClusterName , endpoint	Total size of the storage database file physically allocated in bytes. This metric is experimental and might change in future releases of Kubernetes. Unit: Bytes Meaningful statistics: Sum, Average, Minimum, Maximum
apiserver_request_ total	ClusterName ClusterName , code, verb	The total number of API requests to the Kubernetes API server.
apiserver_request_ duration_seconds	ClusterName ClusterName , verb	Responce latency for API requests to the Kubernetes API server.

Metric name	Dimensions	Description
apiserver_admission_controller_admission_duration_seconds	ClusterName ClusterName , operation	Admission controller latency in seconds. An admission controller is code which intercepts requests to the Kubernetes API server.
rest_client_request_duration_seconds	ClusterName ClusterName , operation	Reponse latency experienced by clients calling the Kubernetes API server. This metric is experimental and may change in future releases of Kubernetes.
rest_client_requests_total	ClusterName ClusterName , code, method	The total number of API requests to the Kubernetes API server made by clients. This metric is experimental and may change in future releases of Kubernetes.
etcd_request_duration_seconds	ClusterName ClusterName , operation	Response latency of API calls to Etcd. This metric is experimental and may change in future releases of Kubernetes.
apiserver_storage_size_bytes	ClusterName ClusterName , endpoint	Size of the storage database file physically allocated in bytes. This metric is experimental and may change in future releases of Kubernetes.
apiserver_longrunning_requests	ClusterName ClusterName , resource	The number of active long-running requests to the Kubernetes API server.

Metric name	Dimensions	Description

`apiserver_current_inflight_requests`	`ClusterName`
	
	`ClusterName` , `request_kind`
		The number of requests that are being processed by Kubernetes API server.

`apiserver_admission_webhook_admission_duration_seconds`	`ClusterName`
	
	`ClusterName` , `name`
		Admission webhook latency in seconds. Admission webhooks are HTTP callbacks that receive admission requests and do something with them.

`apiserver_admission_step_admission_duration_seconds`	`ClusterName`
	
	`ClusterName` , `operation`
		Admission sub-step latency in seconds.

`apiserver_requested_deprecated_apis`	`ClusterName`
	
	`ClusterName` , `group`
		Number of requests to deprecated APIs on the Kubernetes API server.

`apiserver_request_total_5xx`	`ClusterName`
	
	`ClusterName` , `code`, `verb`
		Number of requests to the Kubernetes API server which were responded to with a 5XX HTTP response code.

`apiserver_storage_list_duration_seconds`	`ClusterName`
	
	`ClusterName` , `resource`
		Response latency of listing objects from Etc. This metric is experimental and may change in future releases of Kubernetes.

`apiserver_flowcontrol_request_concurrency_limit`	`ClusterName`
	
	`ClusterName` , `priority_level`
		The number of threads used by the currently executing requests in the API Priority and Fairness subsystem.

Metric name	Dimensions	Description

`apiserver_flowcontrol_rejected_requests_total`	`ClusterName`
	
	`ClusterName` , `reason`
		Number of requests rejected by API Priority and Fairness subsystem. This metric is experimental and may change in future releases of Kubernetes.

`apiserver_current_inqueue_requests`	`ClusterName`
	
	`ClusterName` , `request_kind`
		The number queued requests queued by the Kubernetes API server. This metric is experimental and may change in future releases of Kubernetes.

NVIDIA GPU metrics

Beginning with version `1.300034.0` of the CloudWatch agent, Container Insights with enhanced observability for Amazon EKS collects NVIDIA GPU metrics from EKS workloads by default. The CloudWatch agent must be installed using the CloudWatch Observability EKS add-on version `v1.3.0-eksbuild.1` or later. For more information, see [Install the CloudWatch agent with the Amazon CloudWatch Observability EKS add-on or the Helm chart](). These NVIDIA GPU metrics that are collected are listed in the table in this section.

For Container Insights to collect NVIDIA GPU metrics, you must meet the following prerequisites:

- You must be using Container Insights with enhanced observability for Amazon EKS, with the Amazon CloudWatch Observability EKS add-on version `v1.3.0-eksbuild.1` or later.
- [The NVIDIA device plugin for Kubernetes]() must be installed in the cluster.
- [The NVIDIA container toolkit]() must be installed on the nodes of the cluster. For example, the Amazon EKS optimized accelerated AMIs are built with the necessary components.

You can opt out of collecting NVIDIA GPU metrics by setting the `accelerated_compute_metrics` option in the beginn CloudWatch agent configuration file to `false`. For more information and an example opt-out configuration, see [(Optional) Additional configuration]().

Metric name	Dimensions	Description
container_gpu_memory_total	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , GpuDevice	The total frame buffer size, in bytes, on the GPU(s) allocated to the container.
container_gpu_memory_used	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , GpuDevice	The bytes of frame buffer used on the GPU(s) allocated to the container.
container_gpu_memory_utilization	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName	The percentage of frame buffer used of the GPU(s) allocated to the container.

Metric name	Dimensions	Description
	ClusterName , Namespace , PodName, FullPodName , ContainerName , GpuDevice	
container_gpu_power_draw	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , GpuDevice	The power usage in watts of the GPU(s) allocated to the container.
container_gpu_temperature	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , GpuDevice	The temperature in degrees celsius of the GPU(s) allocated to the container.

Metric name	Dimensions	Description
container_gpu_utilization	ClusterName ClusterName ,Namespace , PodName, ContainerName ClusterName ,Namespace ,PodName, FullPodName , ContainerName ClusterName ,Namespace ,PodName, FullPodName , ContainerName ,GpuDevice	The percentage utilization of the GPU(s) allocated to the container.
node_gpu_memory_total	ClusterName ClusterName ,InstanceId , NodeName ClusterName ,InstanceId , InstanceType ,NodeName, GpuDevice	The total frame buffer size, in bytes, on the GPU(s) allocated to the node.
node_gpu_memory_used	ClusterName ClusterName ,InstanceId , NodeName ClusterName ,InstanceId , InstanceType ,NodeName, GpuDevice	The bytes of frame buffer used on the GPU(s) allocated to the node.

Metric name	Dimensions	Description
node_gpu_memory_utilization	ClusterName ClusterName , InstanceId , NodeName ClusterName , InstanceId , InstanceType , NodeName, GpuDevice	The percentage of frame buffer used on the GPU(s) allocated to the node.
node_gpu_power_draw	ClusterName ClusterName , InstanceId , NodeName ClusterName , InstanceId , InstanceType , NodeName, GpuDevice	The power usage in watts of the GPU(s) allocated to the node.
node_gpu_temperature	ClusterName ClusterName , InstanceId , NodeName ClusterName , InstanceId , InstanceType , NodeName, GpuDevice	The temperature in degrees celsius of the GPU(s) allocated to the node.
node_gpu_utilization	ClusterName ClusterName , InstanceId , NodeName ClusterName , InstanceId , InstanceType , NodeName, GpuDevice	The percentage utilization of the GPU(s) allocated to the node.

Metric name	Dimensions	Description
pod_gpu_memory_total	ClusterName ClusterName , Namespace ClusterName , Namespace , Service ClusterName , Namespace , PodName ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , PodName, FullPodName . GpuDevice	The total frame buffer size, in bytes, on the GPU(s) allocated to the pod.
pod_gpu_memory_used	ClusterName ClusterName , Namespace ClusterName , Namespace , Service ClusterName , Namespace , PodName ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , PodName, FullPodName . GpuDevice	The bytes of frame buffer used on the GPU(s) allocated to the pod.

Metric name	Dimensions	Description
pod_gpu_memory_utilization	ClusterName ClusterName ,Namespace ClusterName ,Namespace , Service ClusterName ,Namespace , PodName ClusterName ,Namespace , PodName, FullPodName ClusterName ,Namespace , PodName, FullPodName . GpuDevice	The percentage of frame buffer used of the GPU(s) allocated to the pod.
pod_gpu_power_draw	ClusterName ClusterName ,Namespace ClusterName ,Namespace , Service ClusterName ,Namespace , PodName ClusterName ,Namespace , PodName, FullPodName ClusterName ,Namespace , PodName, FullPodName . GpuDevice	The power usage in watts of the GPU(s) allocated to the pod.

Metric name	Dimensions	Description

`pod_gpu_temperature`	`ClusterName`
	
	`ClusterName` , `Namespace`
	
	`ClusterName` , `Namespace` , `Service`
	
	`ClusterName` , `Namespace` , `PodName`
	
	`ClusterName` , `Namespace` , `PodName`, `FullPodName`
	
	`ClusterName` , `Namespace` , `PodName`, `FullPodName` . `GpuDevice`
		The temperature in degrees Celsius of the GPU(s) allocated to the pod.

`pod_gpu_utilization`	`ClusterName`
	
	`ClusterName` , `Namespace` , `PodName`, `ContainerName`
	
	`ClusterName` , `Namespace` , `PodName`, `FullPodName` , `ContainerName`
	
	`ClusterName` , `Namespace` , `PodName`, `FullPodName` , `ContainerName` , `GpuDevice`
		The percentage utilization of the GPU(s) allocated to the pod.

Amazon Neuron metrics for Amazon Trainium and Amazon Inferentia

Beginning with version `1.300036.0` of the CloudWatch agent, Container Insights with enhanced observability for Amazon EKS collects accelerated computing metrics from Amazon Trainium and Amazon Inferentia accelerators by default. The CloudWatch agent must be installed using the CloudWatch Observability EKS add-on version `v1.5.0-eksbuild.1` or later. For more information about the add-on, see Install the CloudWatch agent with the Amazon CloudWatch

Observability EKS add-on or the Helm chart. For more information about Amazon Trainium, see Amazon Trainium. For more information about Amazon Inferentia, see Amazon Inferentia.

For Container Insights to collect Amazon Neuron metrics, you must meet the following prerequisites:

- You must be using Container Insights with enhanced observability for Amazon EKS, with the Amazon CloudWatch Observability EKS add-on version `v1.5.0-eksbuild.1` or later.
- The Neuron driver must be installed on the nodes of the cluster.
- The Neuron device plugin must be installed on the cluster. For example, the Amazon EKS optimized accelerated AMIs are built with the necessary components.

The metrics that are collected are listed in the table in this section. The metrics are collected for Amazon Trainium, Amazon Inferentia, and Amazon Inferentia2.

The CloudWatch agent collects these metrics from the Neuron monitor and does the necessary Kubernetes resource correlation to deliver metrics at the pod and container levels

Metric name	Dimensions	Description

`container_neuronco re_utiliz ation`	`ClusterName`
	
	`ClusterName` , `Namespace` , `PodName`, `ContainerName`
	
	`ClusterName` , `Namespace` , `PodName`, `FullPodName` , `ContainerName`
	
	`ClusterName` , `Namespace` , `PodName`, `FullPodName` , `ContainerName` , `NeuronDev ice` , `NeuronCore`
		NeuronCore utilization, during the captured period of the NeuronCore allocated to the container.
		
		Unit: Percent

`container_neuronco re_memory`	`ClusterName`
	
	`ClusterName` , `Namespace` , `PodName`, `ContainerName`
		The amount of device memory used for constants during training by the NeuronCore that is allocated

Metric name	Dimensions	Description
_usage_constants	ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , NeuronDevice , NeuronCore	to the container (or weights during inference). Unit: Bytes
container_neuroncore_memory_usage_model_code	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , NeuronDevice , NeuronCore	The amount of device memory used for the models' executable code by the NeuronCore that is allocated to the container. Unit: Bytes
container_neuroncore_memory_usage_model_shared_scratchpad	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , NeuronDevice , NeuronCore	The amount of device memory used for the scratchpad shared of the models by the NeuronCore that is allocated to the container. This memory region is reserved for the models. Unit: Bytes

Metric name	Dimensions	Description

`container_neuroncore_memory_usage_runtime_memory`	`ClusterName`
	
	`ClusterName` , `Namespace` , `PodName`, `ContainerName`
	
	`ClusterName` , `Namespace` , `PodName`, `FullPodName` , `ContainerName`
	
	`ClusterName` , `Namespace` , `PodName`, `FullPodName` , `ContainerName` , `NeuronDevice` , `NeuronCore`
		The amount of device memory used for the Neuron runtime by the NeuronCore allocated to the container.
		
		Unit: Bytes

`container_neuroncore_memory_usage_tensors`	`ClusterName`
	
	`ClusterName` , `Namespace` , `PodName`, `ContainerName`
	
	`ClusterName` , `Namespace` , `PodName`, `FullPodName` , `ContainerName`
	
	`ClusterName` , `Namespace` , `PodName`, `FullPodName` , `ContainerName` , `NeuronDevice` , `NeuronCore`
		The amount of device memory used for tensors by the NeuronCore allocated to the container.
		
		Unit: Bytes

Metric name	Dimensions	Description
container_neuroncore_memory_usage_total	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , NeuronDevice , NeuronCore	The total amount of memory used by the NeuronCore allocated to the container. Unit: Bytes
container_neurondevice_hw_ecc_events_total	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , NeuronDevice	The number of corrected and uncorrected ECC events for the on-chip SRAM and device memory of the Neuron device on the node. Unit: Count

Metric name	Dimensions	Description
pod_neuroncore_utilization	ClusterName ClusterName , Namespace ClusterName , Namespace , Service ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , PodName, FullPodName , NeuronDevice , NeuronCore	The NeuronCore utilization during the captured period of the NeuronCore allocated to the pod. Unit: Percent
pod_neuroncore_memory_usage_constants	ClusterName ClusterName , Namespace ClusterName , Namespace , Service ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , PodName, FullPodName , NeuronDevice , NeuronCore	The amount of device memory used for constants during training by the NeuronCore that is allocated to the pod (or weights during inference). Unit: Bytes

Metric name	Dimensions	Description
pod_neuroncore_memory_usage_model_code	ClusterName ClusterName ,Namespace ClusterName ,Namespace , Service ClusterName ,Namespace , PodName, FullPodName ClusterName ,Namespace , PodName, FullPodName , NeuronDevice ,NeuronCore	The amount of device memory used for the models' executable code by the NeuronCore that is allocated to the pod. Unit: Bytes
pod_neuroncore_memory_usage_model_shared_scratchpad	ClusterName ClusterName ,Namespace ClusterName ,Namespace , Service ClusterName ,Namespace , PodName, FullPodName ClusterName ,Namespace , PodName, FullPodName , NeuronDevice ,NeuronCore	The amount of device memory used for the scratchpad shared of the models by the NeuronCore that is allocated to the pod. This memory region is reserved for the models. Unit: Bytes

Metric name	Dimensions	Description
pod_neuroncore_memory_usage_runtime_memory	ClusterName ClusterName ,Namespace ClusterName ,Namespace , Service ClusterName ,Namespace , PodName, FullPodName ClusterName ,Namespace , PodName, FullPodName , NeuronDevice ,NeuronCore	The amount of device memory used for the Neuron runtime by the NeuronCore allocated to the pod. Unit: Bytes
pod_neuroncore_memory_usage_tensors	ClusterName ClusterName ,Namespace ClusterName ,Namespace , Service ClusterName ,Namespace , PodName, FullPodName ClusterName ,Namespace , PodName, FullPodName , NeuronDevice ,NeuronCore	The amount of device memory used for tensors by the NeuronCore allocated to the pod. Unit: Bytes

Metric name	Dimensions	Description
pod_neuroncore_memory_usage_total	ClusterName ClusterName , Namespace ClusterName , Namespace , Service ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , PodName, FullPodName , NeuronDevice , NeuronCore	The total amount of memory used by the NeuronCore allocated to the pod. Unit: Bytes
pod_neurondevice_hw_ecc_events_total	ClusterName ClusterName , Namespace ClusterName , Namespace , Service ClusterName , Namespace , PodName ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , PodName, FullPodName , NeuronDevice	The number of corrected and uncorrected ECC events for the on-chip SRAM and device memory of the Neuron device allocated to a pod. Unit: Bytes

Metric name	Dimensions	Description

`node_neuroncore_utilization`	`ClusterName`
	
	`ClusterName` , `InstanceId` , `NodeName`
	
	`ClusterName` , `InstanceType` , `InstanceId` , `NodeName`, `NeuronDevice` , `NeuronCore`
		The NeuronCore utilization during the captured period of the NeuronCore allocated to the node.
		
		Unit: Percent

`node_neuroncore_memory_usage_constants`	`ClusterName`
	
	`ClusterName` , `InstanceId` , `NodeName`
	
	`ClusterName` , `InstanceType` , `InstanceId` , `NodeName`, `NeuronDevice` , `NeuronCore`
		The amount of device memory used for constants during training by the NeuronCore that is allocated to the node (or weights during inference).
		
		Unit: Bytes

`node_neuroncore_memory_usage_model_code`	`ClusterName`
	
	`ClusterName` , `InstanceId` , `NodeName`
	
	`ClusterName` , `InstanceType` , `InstanceId` , `NodeName`, `NeuronDevice` , `NeuronCore`
		The amount of device memory used for models' executable code by the NeuronCore that is allocated to the node.
		
		Unit: Bytes

`node_neuroncore_memory_usage_model_shared_scratchpad`	`ClusterName`
	
	`ClusterName` , `InstanceId` , `NodeName`
	
	`ClusterName` , `InstanceType` , `InstanceId` , `NodeName`, `NeuronDevice` , `NeuronCore`
		The amount of device memory used for the scratchpad shared of the models by the NeuronCore that is allocated to the node. This is a memory region reserved for the models.
		
		Unit: Bytes

Metric name	Dimensions	Description
node_neuroncore_memory_usage_runtime_memory	ClusterName ClusterName , InstanceId , NodeName ClusterName , InstanceType , InstanceId , NodeName, NeuronDevice , NeuronCore	The amount of device memory used for the Neuron runtime by the NeuronCore that is allocated to the node. Unit: Bytes
node_neuroncore_memory_usage_tensors	ClusterName ClusterName , InstanceId , NodeName ClusterName , InstanceType , InstanceId , NodeName, NeuronDevice , NeuronCore	The amount of device memory used for tensors by the NeuronCore that is allocated to the node. Unit: Bytes
node_neuroncore_memory_usage_total	ClusterName ClusterName , InstanceId , NodeName ClusterName , InstanceType , InstanceId , NodeName, NeuronDevice , NeuronCore	The total amount of memory used by the NeuronCore that is allocated to the node. Unit: Bytes
node_neuron_execution_errors_total	ClusterName ClusterName , InstanceId , NodeName	The total number of execution errors on the node. This is calculated by the CloudWatch agent by aggregating the errors of the following types: generic, numerical , transient , model, runtime, and hardware Unit: Count

Metric name	Dimensions	Description
node_neurondevice_runtime_memory_used_bytes	ClusterName ClusterName , InstanceId , NodeName	The total Neuron device memory usage in bytes on the node. Unit: Bytes
node_neuron_execution_latency	ClusterName ClusterName , InstanceId , NodeName	In seconds, the latency for an execution on the node as measured by the Neuron runtime. Unit: Seconds
node_neurondevice_hw_ecc_events_total	ClusterName ClusterName , InstanceId , NodeName ClusterName , InstanceId , NodeName, NeuronDevice	The number of corrected and uncorrected ECC events for the on-chip SRAM and device memory of the Neuron device on the node. Unit: Count

Amazon Elastic Fabric Adapter (EFA) metrics

Beginning with version `1.300037.0` of the CloudWatch agent, Container Insights with enhanced observability for Amazon EKS collects Amazon Elastic Fabric Adapter (EFA) metrics from Amazon EKS clusters on Linux instances. The CloudWatch agent must be installed using the CloudWatch Observability EKS add-on version `v1.5.2-eksbuild.1` or later. For more information about the add-on, see Install the CloudWatch agent with the Amazon CloudWatch Observability EKS add-on or the Helm chart. For more information about Amazon Elastic Fabric Adapter, see Elastic Fabric Adapter.

For Container Insights to collect Amazon Elastic Fabric adapter metrics, you must meet the following prerequisites:

- You must be using Container Insights with enhanced observability for Amazon EKS, with the Amazon CloudWatch Observability EKS add-on version `v1.5.2-eksbuild.1` or later.

- The EFA device plugin must be installed on the cluster. For more information, see [aws-efa-k8s-device-plugin](https://github.com/aws-samples/aws-efa-eks) on GitHub.

The metrics that are collected are listed in the following table.

Metric name	Dimensions	Description

`container_efa_rx_bytes`	`ClusterName`
	
	`ClusterName`, `Namespace`, `PodName`, `ContainerName`
	
	`ClusterName`, `Namespace`, `PodName`, `FullPodName`, `ContainerName`
	
	`ClusterName`, `Namespace`, `PodName`, `FullPodName`, `ContainerName`, `NetworkInterfaceId`
		The number of bytes per second received by the EFA device(s) allocated to the container.
		
		Unit: Bytes/Second

`container_efa_tx_bytes`	`ClusterName`
	
	`ClusterName`, `Namespace`, `PodName`, `ContainerName`
	
	`ClusterName`, `Namespace`, `PodName`, `FullPodName`, `ContainerName`
	
	`ClusterName`, `Namespace`, `PodName`, `FullPodName`, `ContainerName`, `NetworkInterfaceId`
		The number of bytes per second transmitted by the EFA device(s) allocated to the container.
		
		Unit: Bytes/Second

`container_efa_rx_dropped`	`ClusterName`
	
	`ClusterName`, `Namespace`, `PodName`, `ContainerName`
		The number of packets that were received and then dropped by the EFA device(s) allocated to the container.

Metric name	Dimensions	Description
	ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , NetworkInterfaceId	Unit: Count/Second
container_efa_rdma_read_bytes	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , NetworkInterfaceId	The number of bytes per second received using remote direct memory access read operations by the EFA device(s) allocated to the container. Unit: Bytes/Second
container_efa_rdma_write_bytes	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , NetworkInterfaceId	The number of bytes per second transmitted using remote direct memory access read operations by the EFA device(s) allocated to the container. Unit: Bytes/Second

Metric name	Dimensions	Description
container_efa_rdma_write_recv_bytes	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , NetworkInterfaceId	The number of bytes per second received during remote direct memory access write operations by the EFA device(s) allocated to the container. Unit: Bytes/Second
pod_efa_rx_bytes	ClusterName ClusterName , Namespace ClusterName , Namespace , Service ClusterName , Namespace , PodName ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , PodName, FullPodName , NetworkInterfaceId	The number of bytes per second received by the EFA device(s) allocated to the pod. Unit: Bytes/Second

Metric name	Dimensions	Description
pod_efa_tx_bytes	ClusterName ClusterName ,Namespace ClusterName ,Namespace , Service ClusterName ,Namespace , PodName ClusterName ,Namespace , PodName, FullPodName ClusterName ,Namespace , PodName, FullPodName , NetworkInterfaceId	The number of bytes per second transmitted by the EFA device(s) allocated to the pod. Unit: Bytes/Second
pod_efa_rx_dropped	ClusterName ClusterName ,Namespace ClusterName ,Namespace , Service ClusterName ,Namespace , PodName ClusterName ,Namespace , PodName, FullPodName ClusterName ,Namespace , PodName, FullPodName , NetworkInterfaceId	The number of packets that were received and then dropped by the EFA device(s) allocated to the pod. Unit: Count/Second

Metric name	Dimensions	Description
pod_efa_rdma_read_bytes	ClusterName ClusterName ,Namespace ClusterName ,Namespace , Service ClusterName ,Namespace , PodName ClusterName ,Namespace , PodName, FullPodName ClusterName ,Namespace , PodName, FullPodName , NetworkInterfaceId	The number of bytes per second received using remote direct memory access read operations by the EFA device(s) allocated to the pod. Unit: Bytes/Second
pod_efa_rdma_write_bytes	ClusterName ClusterName ,Namespace ClusterName ,Namespace , Service ClusterName ,Namespace , PodName ClusterName ,Namespace , PodName, FullPodName ClusterName ,Namespace , PodName, FullPodName , NetworkInterfaceId	The number of bytes per second transmitted using remote direct memory access read operations by the EFA device(s) allocated to the pod. Unit: Bytes/Second

Metric name	Dimensions	Description
pod_efa_rdma_write_recv_bytes	ClusterName ClusterName , Namespace ClusterName , Namespace , Service ClusterName , Namespace , PodName ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , PodName, FullPodName , NetworkInterfaceId	The number of bytes per second received during remote direct memory access write operations by the EFA device(s) allocated to the pod. Unit: Bytes/Second
node_efa_rx_bytes	ClusterName ClusterName , InstanceId , NodeName ClusterName , InstanceId , InstanceType , NodeName, NetworkInterfaceId	The number of bytes per second received by the EFA device(s) allocated to the node. Unit: Bytes/Second
node_efa_tx_bytes	ClusterName ClusterName , InstanceId , NodeName ClusterName , InstanceId , InstanceType , NodeName, NetworkInterfaceId	The number of bytes per second transmitted by the EFA device(s) allocated to the node. Unit: Bytes/Second

Metric name	Dimensions	Description
node_efa_rx_dropped	ClusterName ClusterName , InstanceId , NodeName ClusterName , InstanceId , InstanceType , NodeName, NetworkInterfaceId	The number of packets that were received and then dropped by the EFA device(s) allocated to the node. Unit: Count/Second
node_efa_rdma_read_bytes	ClusterName ClusterName , InstanceId , NodeName ClusterName , InstanceId , InstanceType , NodeName, NetworkInterfaceId	The number of bytes per second received using remote direct memory access read operations by the EFA device(s) allocated to the node. Unit: Bytes/Second
node_efa_rdma_write_bytes	ClusterName ClusterName , InstanceId , NodeName ClusterName , InstanceId , InstanceType , NodeName, NetworkInterfaceId	The number of bytes per second transmitted using remote direct memory access read operations by the EFA device(s) allocated to the pod. Unit: Bytes/Second
node_efa_rdma_write_recv_bytes	ClusterName ClusterName , InstanceId , NodeName ClusterName , InstanceId , InstanceType , NodeName, NetworkInterfaceId	The number of bytes per second received during remote direct memory access write operations by the EFA device(s) allocated to the node. Unit: Bytes/Second

Amazon SageMaker AI HyperPod metrics

Beginning with version `v2.0.1-eksbuild.1` of the the CloudWatch Observability EKS add-on, Container Insights with enhanced observability for Amazon EKS automatically collects Amazon SageMaker AI HyperPod metrics from Amazon EKS clusters. For more information about the add-on, see Install the CloudWatch agent with the Amazon CloudWatch Observability EKS add-on or the Helm chart. For more information about Amazon SageMaker AI HyperPod, see Amazon SageMaker AI HyperPod.

The metrics that are collected are listed in the following table.

Metric name	Dimensions	Description

`hyperpod_node_health_status_unschedulable`	`ClusterName`
	
	`ClusterName` , `InstanceId` , `NodeName`
		Indicates if a node is labeled as `Unschedulable` by Amazon SageMaker AI HyperPod. This means that the node is running deep health checks and is not available for running workloads.
		
		Unit: Count

`hyperpod_node_health_status_schedulable`	`ClusterName`
	
	`ClusterName` , `InstanceId` , `NodeName`
		Indicates if a node is labeled as `Schedulable` by Amazon SageMaker AI HyperPod. This means that the node has passed basic health checks or deep health checks and is available for running workloads.
		
		Unit: Count

`hyperpod_node_health_status_unschedulable_pen`	`ClusterName`
	
	`ClusterName` , `InstanceId` , `NodeName`
		Indicates if a node is labeled as `UnschedulablePendingReplacement` by HyperPod. This means that the node has failed deep health checks or health monitoring agent checks and requires a replacement.

Metric name	Dimensions	Description
ding_repl acement		If automatic node recovery is enabled, the node will be automatic ally replaced by Amazon SageMaker AI HyperPod. Unit: Count
hyperpod_ node_heal th_status _unschedu lable_pen ding_rebo ot	ClusterName ClusterName , InstanceId , NodeName	Indicates if a node is labeled as UnschedulablePendi ngReboot by Amazon SageMaker AI HyperPod. This means that the node is running deep health checks and requires a reboot. If automatic node recovery is enabled, the node will be automatic ally rebooted by Amazon SageMaker AI HyperPod. Unit: Count

Amazon EKS and Kubernetes Container Insights metrics

The following tables list the metrics and dimensions that Container Insights collects for Amazon EKS and Kubernetes. These metrics are in the ContainerInsights namespace. For more information, see Metrics.

If you do not see any Container Insights metrics in your console, be sure that you have completed the setup of Container Insights. Metrics do not appear before Container Insights has been set up completely. For more information, see Setting up Container Insights.

Metric name	Dimensions	Description
cluster_failed_nod e_count	ClusterName	The number of failed worker nodes in the cluster. A node is considered failed if it is

Metric name	Dimensions	Description
		suffering from any *node conditions*. For more information, see Conditions in the Kubernetes documentation.
`cluster_node_count`	`ClusterName`	The total number of worker nodes in the cluster.

`namespace_number_of_running_pods`	`Namespace ClusterName`
	
	`ClusterName`
		The number of pods running per namespace in the resource that is specified by the dimensions that you're using.

`node_cpu_limit`	`ClusterName`	The maximum number of CPU units that can be assigned to a single node in this cluster.

Metric name	Dimensions	Description

`node_cpu_reserved_capacity`	`NodeName`, `ClusterName` , `InstanceId`
	
	`ClusterName`
		The percentage of CPU units that are reserved for node components, such as kubelet, kube-proxy, and Docker.
		
		Formula: `node_cpu_request / node_cpu_limit`
		
		Note
		`node_cpu_request` is not reported directly as a metric, but is a field in performance log events. For more information, see [Relevant fields in performance log events for Amazon EKS and Kubernetes]().

`node_cpu_usage_total`	`ClusterName`	The number of CPU units being used on the nodes in the cluster.

`node_cpu_utilization`	`NodeName`, `ClusterName` , `InstanceId`
	
	`ClusterName`
		The total percentage of CPU units being used on the nodes in the cluster.
		
		Formula: `node_cpu_usage_total / node_cpu_limit`

Metric name	Dimensions	Description

`node_gpu_limit`	`ClusterName`
	
	`ClusterName` , `InstanceId` , `NodeName`
		The total number of GPU(s) available on the node.

`node_gpu_usage_total`	`ClusterName`
	
	`ClusterName` , `InstanceId` , `NodeName`
		The number of GPU(s) being used by the running pods on the node.

`node_gpu_reserved_capacity`	`ClusterName`
	
	`ClusterName` , `InstanceId` , `NodeName`
		The percentage of GPU currently being reserved on the node. The formula is, `node_gpu_request / node_gpu_limit` .
		
		Note
		`node_gpu_request` is not reported directly as a metric, but is a field in performance log events. For more information, see [Relevant fields in performance log events for Amazon EKS and Kubernetes]().

Metric name	Dimensions	Description

`node_filesystem_utilization`	`NodeName`, `ClusterName` , `InstanceId`
	
	`ClusterName`
		The total percentage of file system capacity being used on nodes in the cluster.
		
		Formula: `node_filesystem_usage / node_filesystem_capacity`
		
		Note
		`node_filesystem_usage` and `node_filesystem_capacity` are not reported directly as metrics, but are fields in performance log events. For more information, see Relevant fields in performance log events for Amazon EKS and Kubernetes.

`node_memory_limit`	`ClusterName`	The maximum amount of memory, in bytes, that can be assigned to a single node in this cluster.

Metric name	Dimensions	Description
node_memory_reserved_capacity	NodeName, ClusterName , InstanceId ClusterName	The percentage of memory currently being used on the nodes in the cluster. Formula: node_memory_request / node_memory_limit **Note** node_memory_request is not reported directly as a metric, but is a field in performance log events. For more information, see Relevant fields in performance log events for Amazon EKS and Kubernetes.
node_memory_utilization	NodeName, ClusterName , InstanceId ClusterName	The percentage of memory currently being used by the node or nodes. It is the percentage of node memory usage divided by the node memory limitation. Formula: node_memory_working_set / node_memory_limit .

Metric name	Dimensions	Description
node_memory_working_set	ClusterName	The amount of memory, in bytes, being used in the working set of the nodes in the cluster.
node_network_total_bytes	NodeName, ClusterName , InstanceId ClusterName	The total number of bytes per second transmitted and received over the network per node in a cluster. Formula: node_network_rx_bytes + node_network_tx_bytes **Note** node_network_rx_bytes and node_network_tx_bytes are not reported directly as metrics, but are fields in performance log events. For more information, see [Relevant fields in performance log events for Amazon EKS and Kubernetes]().
node_number_of_running_containers	NodeName, ClusterName , InstanceId ClusterName	The number of running containers per node in a cluster.

Metric name	Dimensions	Description
node_number_of_running_pods	NodeName, ClusterName , InstanceId ClusterName	The number of running pods per node in a cluster.
pod_cpu_reserved_capacity	PodName, Namespace , ClusterName ClusterName	The CPU capacity that is reserved per pod in a cluster. Formula: pod_cpu_request / node_cpu_limit **Note** pod_cpu_request is not reported directly as a metric, but is a field in performance log events. For more information, see [Relevant fields in performance log events for Amazon EKS and Kubernetes].
pod_cpu_utilization	PodName, Namespace , ClusterName Namespace , ClusterName Service, Namespace , ClusterName ClusterName	The percentage of CPU units being used by pods. Formula: pod_cpu_usage_total / node_cpu_limit

Metric name	Dimensions	Description
pod_cpu_utilization_over_pod_limit	PodName, Namespace , ClusterName Namespace , ClusterName Service, Namespace , ClusterName ClusterName	The percentage of CPU units being used by pods relative to the pod limit. Formula: pod_cpu_usage_total / pod_cpu_limit
pod_gpu_request	ClusterName ClusterName , Namespace , PodName ClusterName , FullPodName , Namespace , PodName	The GPU requests for the pod. This value must always be equal to pod_gpu_limit .
pod_gpu_limit	ClusterName ClusterName , Namespace , PodName ClusterName , FullPodName , Namespace , PodName	The maximum number of GPU(s) that can be assigned to the pod in a node.
pod_gpu_usage_total	ClusterName ClusterName , Namespace , PodName ClusterName , FullPodName , Namespace , PodName	The number of GPU(s) being allocated on the pod.

Metric name	Dimensions	Description

`pod_gpu_reserved_capacity`	`ClusterName`
	
	`ClusterName` , `Namespace` , `PodName`
	
	`ClusterName` , `FullPodName` , `Namespace` , `PodName`
		The percentage of GPU currently being reserved for the pod. The formula is - pod_gpu_request / node_gpu_reserved_capacity.

`pod_memory_reserved_capacity`	`PodName`, `Namespace` , `ClusterName`
	
	`ClusterName`
		The percentage of memory that is reserved for pods.
		
		Formula: `pod_memory_request / node_memory_limit`
		
		Note
		
		`pod_memory_request` is not reported directly as a metric, but is a field in performance log events. For more information, see [Relevant fields in performance log events for Amazon EKS and Kubernetes]().

Metric name	Dimensions	Description
pod_memory_utilization	PodName, Namespace , ClusterName Namespace , ClusterName Service, Namespace , ClusterName ClusterName	The percentage of memory currently being used by the pod or pods. Formula: pod_memory_working_set / node_memory_limit
pod_memory_utilization_over_pod_limit	PodName, Namespace , ClusterName Namespace , ClusterName Service, Namespace , ClusterName ClusterName	The percentage of memory that is being used by pods relative to the pod limit. If any containers in the pod don't have a memory limit defined, this metric doesn't appear. Formula: pod_memory_working_set / pod_memory_limit

Metric name	Dimensions	Description

`pod_network_rx_bytes`	`PodName`, `Namespace` , `ClusterName`
	
	`Namespace` , `ClusterName`
	
	`Service`, `Namespace` , `ClusterName`
	
	`ClusterName`
		The number of bytes per second being received over the network by the pod.
		
		Formula: `sum(pod_interface_network_rx_bytes)`
		
		Note
		
		`pod_interface_network_rx_bytes` is not reported directly as a metric, but is a field in performance log events. For more information, see Relevant fields in performance log events for Amazon EKS and Kubernetes.

Metric name	Dimensions	Description
pod_network_tx_bytes	PodName, Namespace , ClusterName Namespace , ClusterName Service, Namespace , ClusterName ClusterName	The number of bytes per second being transmitted over the network by the pod. Formula: sum(pod_interface_network_tx_bytes) **Note** pod_interface_network_tx_bytes is not reported directly as a metric, but is a field in performance log events. For more information, see Relevant fields in performance log events for Amazon EKS and Kubernetes.
pod_number_of_container_restarts	PodName, Namespace , ClusterName	The total number of container restarts in a pod.
service_number_of_running_pods	Service, Namespace , ClusterName ClusterName	The number of pods running the service or services in the cluster.

Kueue metrics

Beginning with version v2.4.0-eksbuild.1 of the the CloudWatch Observability EKS add-on, Container Insights for Amazon EKS supports collecting Kueue metrics from Amazon EKS clusters.

For more information about the add-on, see [Install the CloudWatch agent with the Amazon CloudWatch Observability EKS add-on or the Helm chart](#).

For information about enabling the metrics, see [Enable Kueue metrics](#) to enable them.

The Kueue metrics that are collected are listed in the following table. These metrics are published into the `ContainerInsights/Prometheus` namespace in CloudWatch. Some of these metrics use the following dimensions:

- `ClusterQueue` is the name of the ClusterQueue
- The possible values of `Status` are `active` and `inadmissible`
- The possible values of `Reason` are `Preempted`, `PodsReadyTimeout`, `AdmissionCheck`, `ClusterQueueStopped`, and `InactiveWorkload`
- `Flavor` is the referenced flavor.
- `Resource` refers to cluster computer resources, such as `cpu`, `memory`, `gpu`, and so on.

Metric name	Dimensions	Description

`kueue_pending_workloads`	`ClusterName` , `ClusterQueue` , `Status`
	
	`ClusterName` , `ClusterQueue`
	
	`ClusterName` , `Status`
	
	`ClusterName`
		The number of pending workloads.

`kueue_evicted_workloads_total`	`ClusterName` , `ClusterQueue` , `Reason`
	
	`ClusterName` , `ClusterQueue`
	
	`ClusterName` , `Reason`
	
	`ClusterName`
		The total number of evicted workloads.

`kueue_admitted_active`	`ClusterName` , `ClusterQueue`
	
	`ClusterName`
		The number of admitted workloads that are active (unsuspended and not finished).

Metric name	Dimensions	Description
ive_workl oads		
kueue_clu ster_queu e_resourc e_usage	ClusterName , ClusterQueue , Resource, Flavor ClusterName , ClusterQueue , Resource ClusterName , ClusterQueue , Flavor ClusterName , ClusterQueue ClusterName	Reports the total resource usage of the ClusterQueue.
kueue_clu ster_queu e_nominal _quota	ClusterName , ClusterQueue , Resource, Flavor ClusterName , ClusterQueue , Resource ClusterName , ClusterQueue , Flavor ClusterName , ClusterQueue ClusterName	Reports the resource quota of the ClusterQueue.

Container Insights performance log reference

This section includes reference information about how Container Insights uses performance log events to collect metrics. When you deploy Container Insights, it automatically creates a log group for the performance log events. You don't need to create this log group yourself.

Topics

- Container Insights performance log events for Amazon ECS

- [Container Insights performance log events for Amazon EKS and Kubernetes]
- [Relevant fields in performance log events for Amazon EKS and Kubernetes]

Container Insights performance log events for Amazon ECS

The following are examples of the performance log events that Container Insights collects from Amazon ECS.

These logs are in CloudWatch Logs, in a log group named `/aws/ecs/containerinsights/CLUSTER_NAME/performance`. Within that log group, each container instance will has a log stream named `AgentTelemetry-CONTAINER_INSTANCE_ID`.

You can query these logs using queries such as `{ $.Type = "Container" }` to view all container log events.

Type: Container


```
{
 "Version":"0",
 "Type":"Container",
 "ContainerName":"sleep",
 "TaskId":"7ac4dfba69214411b4783a3b8189c9ba",
 "TaskDefinitionFamily":"sleep360",
 "TaskDefinitionRevision":"1",
 "ContainerInstanceId":"0d7650e6dec34c1a9200f72098071e8f",
 "EC2InstanceId":"i-0c470579dbcdbd2f3",
 "ClusterName":"MyCluster",
 "Image":"busybox",
 "ContainerKnownStatus":"RUNNING",
 "Timestamp":1623963900000,
 "CpuUtilized":0.0,
 "CpuReserved":10.0,
 "MemoryUtilized":0,
 "MemoryReserved":10,
 "StorageReadBytes":0,
 "StorageWriteBytes":0,
 "NetworkRxBytes":0,
 "NetworkRxDropped":0,
 "NetworkRxErrors":0,
 "NetworkRxPackets":14,
 "NetworkTxBytes":0,
 "NetworkTxDropped":0,
```

```
 "NetworkTxErrors":0,
 "NetworkTxPackets":0
}
```

Type: Task

Even though the units for `StorageReadBytes` and `StorageWriteBytes` are in Bytes/Second, the values represent the cumulative number of bytes read from and written to storage, respectively.


```
{
    "Version": "0",
    "Type": "Task",
    "TaskId": "7ac4dfba69214411b4783a3b8189c9ba",
    "TaskDefinitionFamily": "sleep360",
    "TaskDefinitionRevision": "1",
    "ContainerInstanceId": "0d7650e6dec34c1a9200f72098071e8f",
    "EC2InstanceId": "i-0c470579dbcdbd2f3",
    "ClusterName": "MyCluster",
    "AccountID": "637146863587",
    "Region": "us-west-2",
    "AvailabilityZone": "us-west-2b",
    "KnownStatus": "RUNNING",
    "LaunchType": "EC2",
    "PullStartedAt": 1623963608201,
    "PullStoppedAt": 1623963610065,
    "CreatedAt": 1623963607094,
    "StartedAt": 1623963610382,
    "Timestamp": 1623963900000,
    "CpuUtilized": 0.0,
    "CpuReserved": 10.0,
    "MemoryUtilized": 0,
    "MemoryReserved": 10,
    "StorageReadBytes": 0,
    "StorageWriteBytes": 0,
    "NetworkRxBytes": 0,
    "NetworkRxDropped": 0,
    "NetworkRxErrors": 0,
    "NetworkRxPackets": 14,
    "NetworkTxBytes": 0,
    "NetworkTxDropped": 0,
    "NetworkTxErrors": 0,
    "NetworkTxPackets": 0,
```

```
    "EBSFilesystemUtilized": 10,
    "EBSFilesystemSize": 20,
    "CloudWatchMetrics": [
        {
            "Namespace": "ECS/ContainerInsights",
            "Metrics": [
                {
                    "Name": "CpuUtilized",
                    "Unit": "None"
                },
                {
                    "Name": "CpuReserved",
                    "Unit": "None"
                },
                {
                    "Name": "MemoryUtilized",
                    "Unit": "Megabytes"
                },
                {
                    "Name": "MemoryReserved",
                    "Unit": "Megabytes"
                },
                {
                    "Name": "StorageReadBytes",
                    "Unit": "Bytes/Second"
                },
                {
                    "Name": "StorageWriteBytes",
                    "Unit": "Bytes/Second"
                },
                {
                    "Name": "NetworkRxBytes",
                    "Unit": "Bytes/Second"
                },
                {
                    "Name": "NetworkTxBytes",
                    "Unit": "Bytes/Second"
                },
                {
                    "Name": "EBSFilesystemSize",
                    "Unit": "Gigabytes"
                },
                {
                    "Name": "EBSFilesystemUtilzed",
```

```
                    "Unit": "Gigabytes"
                }
            ],
            "Dimensions": [
                ["ClusterName"],
                [
                    "ClusterName",
                    "TaskDefinitionFamily"
                ]
            ]
        }
    ]
}
```

Type: Service

```
{
    "Version": "0",
    "Type": "Service",
    "ServiceName": "myCIService",
    "ClusterName": "myCICluster",
    "Timestamp": 1561586460000,
    "DesiredTaskCount": 2,
    "RunningTaskCount": 2,
    "PendingTaskCount": 0,
    "DeploymentCount": 1,
    "TaskSetCount": 0,
    "CloudWatchMetrics": [
        {
            "Namespace": "ECS/ContainerInsights",
            "Metrics": [
                {
                    "Name": "DesiredTaskCount",
                    "Unit": "Count"
                },
                {
                    "Name": "RunningTaskCount",
                    "Unit": "Count"
                },
                {
                    "Name": "PendingTaskCount",
                    "Unit": "Count"
                },
```

```
                {
                    "Name": "DeploymentCount",
                    "Unit": "Count"
                },
                {
                    "Name": "TaskSetCount",
                    "Unit": "Count"
                }
            ],
            "Dimensions": [
                [
                    "ServiceName",
                    "ClusterName"
                ]
            ]
        }
    ]
}
```

Type: Volume

```
{
    "Version": "0",
    "Type": "Volume",
    "TaskDefinitionFamily": "myCITaskDef",
    "TaskId": "7ac4dfba69214411b4783a3b8189c9ba",
    "ClusterName": "myCICluster",
    "ServiceName": "myCIService",
    "VolumeId": "vol-1233436545ff708cb",
    "InstanceId": "i-0c470579dbcdbd2f3",
    "LaunchType": "EC2",
    "VolumeName": "MyVolumeName",
    "EBSFilesystemUtilized": 10,
    "EBSFilesystemSize": 20,
    "CloudWatchMetrics": [
        {
            "Namespace": "ECS/ContainerInsights",
            "Metrics": [
                {
                    "Name": "EBSFilesystemSize",
                    "Unit": "Gigabytes"
                },
                {
```

```
                    "Name": "EBSFilesystemUtilzed",
                    "Unit": "Gigabytes"
                }
            ],
            "Dimensions": [
                ["ClusterName"],
                [
                    "VolumeName",
                    "TaskDefinitionFamily",
                    "ClusterName"
                ],
                [
                    "ServiceName",
                    "ClusterName"
                ]
            ]
        }
    ]
}
```

Type: Cluster

```
{
    "Version": "0",
    "Type": "Cluster",
    "ClusterName": "myCICluster",
    "Timestamp": 1561587300000,
    "TaskCount": 5,
    "ContainerInstanceCount": 5,
    "ServiceCount": 2,
    "CloudWatchMetrics": [
        {
            "Namespace": "ECS/ContainerInsights",
            "Metrics": [
                {
                    "Name": "TaskCount",
                    "Unit": "Count"
                },
                {
                    "Name": "ContainerInstanceCount",
                    "Unit": "Count"
                },
                {
```

```
                    "Name": "ServiceCount",
                    "Unit": "Count"
                }
            ],
            "Dimensions": [
                [
                    "ClusterName"
                ]
            ]
        }
    ]
}
```

Container Insights performance log events for Amazon EKS and Kubernetes

The following are examples of the performance log events that Container Insights collects from Amazon EKS and Kubernetes clusters.

Type: Node

```
{
  "AutoScalingGroupName": "eksctl-myCICluster-nodegroup-standard-workers-
NodeGroup-1174PV2WHZAYU",
  "CloudWatchMetrics": [
    {
      "Metrics": [
        {
          "Unit": "Percent",
          "Name": "node_cpu_utilization"
        },
        {
          "Unit": "Percent",
          "Name": "node_memory_utilization"
        },
        {
          "Unit": "Bytes/Second",
          "Name": "node_network_total_bytes"
        },
        {
          "Unit": "Percent",
          "Name": "node_cpu_reserved_capacity"
        },
        {
```

```
          "Unit": "Percent",
          "Name": "node_memory_reserved_capacity"
        },
        {
          "Unit": "Count",
          "Name": "node_number_of_running_pods"
        },
        {
          "Unit": "Count",
          "Name": "node_number_of_running_containers"
        }
      ],
      "Dimensions": [
        [
          "NodeName",
          "InstanceId",
          "ClusterName"
        ]
      ],
      "Namespace": "ContainerInsights"
    },
    {
      "Metrics": [
        {
          "Unit": "Percent",
          "Name": "node_cpu_utilization"
        },
        {
          "Unit": "Percent",
          "Name": "node_memory_utilization"
        },
        {
          "Unit": "Bytes/Second",
          "Name": "node_network_total_bytes"
        },
        {
          "Unit": "Percent",
          "Name": "node_cpu_reserved_capacity"
        },
        {
          "Unit": "Percent",
          "Name": "node_memory_reserved_capacity"
        },
        {
```

```
          "Unit": "Count",
          "Name": "node_number_of_running_pods"
        },
        {
          "Unit": "Count",
          "Name": "node_number_of_running_containers"
        },
        {
          "Name": "node_cpu_usage_total"
        },
        {
          "Name": "node_cpu_limit"
        },
        {
          "Unit": "Bytes",
          "Name": "node_memory_working_set"
        },
        {
          "Unit": "Bytes",
          "Name": "node_memory_limit"
        }
      ],
      "Dimensions": [
        [
          "ClusterName"
        ]
      ],
      "Namespace": "ContainerInsights"
    }
  ],
  "ClusterName": "myCICluster",
  "InstanceId": "i-1234567890123456",
  "InstanceType": "t3.xlarge",
  "NodeName": "ip-192-0-2-0.us-west-2.compute.internal",
  "Sources": [
    "cadvisor",
    "/proc",
    "pod",
    "calculated"
  ],
  "Timestamp": "1567096682364",
  "Type": "Node",
  "Version": "0",
  "kubernetes": {
```

```
    "host": "ip-192-168-75-26.us-west-2.compute.internal"
  },
  "node_cpu_limit": 4000,
  "node_cpu_request": 1130,
  "node_cpu_reserved_capacity": 28.249999999999996,
  "node_cpu_usage_system": 33.794636630852764,
  "node_cpu_usage_total": 136.47852169244098,
  "node_cpu_usage_user": 71.67075111567326,
  "node_cpu_utilization": 3.4119630423110245,
  "node_memory_cache": 3103297536,
  "node_memory_failcnt": 0,
  "node_memory_hierarchical_pgfault": 0,
  "node_memory_hierarchical_pgmajfault": 0,
  "node_memory_limit": 16624865280,
  "node_memory_mapped_file": 406646784,
  "node_memory_max_usage": 4230746112,
  "node_memory_pgfault": 0,
  "node_memory_pgmajfault": 0,
  "node_memory_request": 1115684864,
  "node_memory_reserved_capacity": 6.7109407818311055,
  "node_memory_rss": 798146560,
  "node_memory_swap": 0,
  "node_memory_usage": 3901444096,
  "node_memory_utilization": 6.601302600149552,
  "node_memory_working_set": 1097457664,
  "node_network_rx_bytes": 35918.392817386324,
  "node_network_rx_dropped": 0,
  "node_network_rx_errors": 0,
  "node_network_rx_packets": 157.67565245448117,
  "node_network_total_bytes": 68264.20276554905,
  "node_network_tx_bytes": 32345.80994816272,
  "node_network_tx_dropped": 0,
  "node_network_tx_errors": 0,
  "node_network_tx_packets": 154.21455923431654,
  "node_number_of_running_containers": 16,
  "node_number_of_running_pods": 13
}
```


Type: NodeFS


```
{
  "AutoScalingGroupName": "eksctl-myCICluster-nodegroup-standard-workers-
NodeGroup-1174PV2WHZAYU",
```

```
"CloudWatchMetrics": [
  {
    "Metrics": [
      {
        "Unit": "Percent",
        "Name": "node_filesystem_utilization"
      }
    ],
    "Dimensions": [
      [
        "NodeName",
        "InstanceId",
        "ClusterName"
      ],
      [
        "ClusterName"
      ]
    ],
    "Namespace": "ContainerInsights"
  }
],
"ClusterName": "myCICluster",
"EBSVolumeId": "aws://us-west-2b/vol-0a53108976d4a2fda",
"InstanceId": "i-1234567890123456",
"InstanceType": "t3.xlarge",
"NodeName": "ip-192-0-2-0.us-west-2.compute.internal",
"Sources": [
  "cadvisor",
  "calculated"
],
"Timestamp": "1567097939726",
"Type": "NodeFS",
"Version": "0",
"device": "/dev/nvme0n1p1",
"fstype": "vfs",
"kubernetes": {
  "host": "ip-192-168-75-26.us-west-2.compute.internal"
},
"node_filesystem_available": 17298395136,
"node_filesystem_capacity": 21462233088,
"node_filesystem_inodes": 10484720,
"node_filesystem_inodes_free": 10367158,
"node_filesystem_usage": 4163837952,
"node_filesystem_utilization": 19.400767547940255
```

```
}
```

Type: NodeDiskIO


```
{
  "AutoScalingGroupName": "eksctl-myCICluster-nodegroup-standard-workers-
NodeGroup-1174PV2WHZAYU",
  "ClusterName": "myCICluster",
  "EBSVolumeId": "aws://us-west-2b/vol-0a53108976d4a2fda",
  "InstanceId": "i-1234567890123456",
  "InstanceType": "t3.xlarge",
  "NodeName": "ip-192-0-2-0.us-west-2.compute.internal",
  "Sources": [
    "cadvisor"
  ],
  "Timestamp": "1567096928131",
  "Type": "NodeDiskIO",
  "Version": "0",
  "device": "/dev/nvme0n1",
  "kubernetes": {
    "host": "ip-192-168-75-26.us-west-2.compute.internal"
  },
  "node_diskio_io_service_bytes_async": 9750.505814277016,
  "node_diskio_io_service_bytes_read": 0,
  "node_diskio_io_service_bytes_sync": 230.6174506688036,
  "node_diskio_io_service_bytes_total": 9981.123264945818,
  "node_diskio_io_service_bytes_write": 9981.123264945818,
  "node_diskio_io_serviced_async": 1.153087253344018,
  "node_diskio_io_serviced_read": 0,
  "node_diskio_io_serviced_sync": 0.03603397666700056,
  "node_diskio_io_serviced_total": 1.1891212300110185,
  "node_diskio_io_serviced_write": 1.1891212300110185
}
```


Type: NodeNet


```
{
  "AutoScalingGroupName": "eksctl-myCICluster-nodegroup-standard-workers-
NodeGroup-1174PV2WHZAYU",
  "ClusterName": "myCICluster",
  "InstanceId": "i-1234567890123456",
  "InstanceType": "t3.xlarge",
  "NodeName": "ip-192-0-2-0.us-west-2.compute.internal",
```

```
  "Sources": [
    "cadvisor",
    "calculated"
  ],
  "Timestamp": "1567096928131",
  "Type": "NodeNet",
  "Version": "0",
  "interface": "eni972f6bfa9a0",
  "kubernetes": {
    "host": "ip-192-168-75-26.us-west-2.compute.internal"
  },
  "node_interface_network_rx_bytes": 3163.008420864309,
  "node_interface_network_rx_dropped": 0,
  "node_interface_network_rx_errors": 0,
  "node_interface_network_rx_packets": 16.575629266820258,
  "node_interface_network_total_bytes": 3518.3935157426017,
  "node_interface_network_tx_bytes": 355.385094878293,
  "node_interface_network_tx_dropped": 0,
  "node_interface_network_tx_errors": 0,
  "node_interface_network_tx_packets": 3.9997714100370625
}
```


Type: Pod


```
{
  "AutoScalingGroupName": "eksctl-myCICluster-nodegroup-standard-workers-
NodeGroup-1174PV2WHZAYU",
  "CloudWatchMetrics": [
    {
      "Metrics": [
        {
          "Unit": "Percent",
          "Name": "pod_cpu_utilization"
        },
        {
          "Unit": "Percent",
          "Name": "pod_memory_utilization"
        },
        {
          "Unit": "Bytes/Second",
          "Name": "pod_network_rx_bytes"
        },
        {
```

```
          "Unit": "Bytes/Second",
          "Name": "pod_network_tx_bytes"
        },
        {
          "Unit": "Percent",
          "Name": "pod_cpu_utilization_over_pod_limit"
        },
        {
          "Unit": "Percent",
          "Name": "pod_memory_utilization_over_pod_limit"
        }
      ],
      "Dimensions": [
        [
          "PodName",
          "Namespace",
          "ClusterName"
        ],
        [
          "Service",
          "Namespace",
          "ClusterName"
        ],
        [
          "Namespace",
          "ClusterName"
        ],
        [
          "ClusterName"
        ]
      ],
      "Namespace": "ContainerInsights"
    },
    {
      "Metrics": [
        {
          "Unit": "Percent",
          "Name": "pod_cpu_reserved_capacity"
        },
        {
          "Unit": "Percent",
          "Name": "pod_memory_reserved_capacity"
        }
      ],
```

```
        "Dimensions": [
          [
            "PodName",
            "Namespace",
            "ClusterName"
          ],
          [
            "ClusterName"
          ]
        ],
        "Namespace": "ContainerInsights"
      },
      {
        "Metrics": [
          {
            "Unit": "Count",
            "Name": "pod_number_of_container_restarts"
          }
        ],
        "Dimensions": [
          [
            "PodName",
            "Namespace",
            "ClusterName"
          ]
        ],
        "Namespace": "ContainerInsights"
      }
    ],
    "ClusterName": "myCICluster",
    "InstanceId": "i-1234567890123456",
    "InstanceType": "t3.xlarge",
    "Namespace": "amazon-cloudwatch",
    "NodeName": "ip-192-0-2-0.us-west-2.compute.internal",
    "PodName": "cloudwatch-agent-statsd",
    "Service": "cloudwatch-agent-statsd",
    "Sources": [
      "cadvisor",
      "pod",
      "calculated"
    ],
    "Timestamp": "1567097351092",
    "Type": "Pod",
    "Version": "0",
```

```
  "kubernetes": {
    "host": "ip-192-168-75-26.us-west-2.compute.internal",
    "labels": {
      "app": "cloudwatch-agent-statsd",
      "pod-template-hash": "df44f855f"
    },
    "namespace_name": "amazon-cloudwatch",
    "pod_id": "2f4ff5ac-c813-11e9-a31d-06e9dde32928",
    "pod_name": "cloudwatch-agent-statsd-df44f855f-ts4q2",
    "pod_owners": [
      {
        "owner_kind": "Deployment",
        "owner_name": "cloudwatch-agent-statsd"
      }
    ],
    "service_name": "cloudwatch-agent-statsd"
  },
  "pod_cpu_limit": 200,
  "pod_cpu_request": 200,
  "pod_cpu_reserved_capacity": 5,
  "pod_cpu_usage_system": 1.4504841104992765,
  "pod_cpu_usage_total": 5.817016867430125,
  "pod_cpu_usage_user": 1.1281543081661038,
  "pod_cpu_utilization": 0.14542542168575312,
  "pod_cpu_utilization_over_pod_limit": 2.9085084337150624,
  "pod_memory_cache": 8192,
  "pod_memory_failcnt": 0,
  "pod_memory_hierarchical_pgfault": 0,
  "pod_memory_hierarchical_pgmajfault": 0,
  "pod_memory_limit": 104857600,
  "pod_memory_mapped_file": 0,
  "pod_memory_max_usage": 25268224,
  "pod_memory_pgfault": 0,
  "pod_memory_pgmajfault": 0,
  "pod_memory_request": 104857600,
  "pod_memory_reserved_capacity": 0.6307275170893897,
  "pod_memory_rss": 22777856,
  "pod_memory_swap": 0,
  "pod_memory_usage": 25141248,
  "pod_memory_utilization": 0.10988455961791709,
  "pod_memory_utilization_over_pod_limit": 17.421875,
  "pod_memory_working_set": 18268160,
  "pod_network_rx_bytes": 9880.697124714186,
  "pod_network_rx_dropped": 0,
```

```
  "pod_network_rx_errors": 0,
  "pod_network_rx_packets": 107.80005532263283,
  "pod_network_total_bytes": 10158.829201483635,
  "pod_network_tx_bytes": 278.13207676944796,
  "pod_network_tx_dropped": 0,
  "pod_network_tx_errors": 0,
  "pod_network_tx_packets": 1.146027574644318,
  "pod_number_of_container_restarts": 0,
  "pod_number_of_containers": 1,
  "pod_number_of_running_containers": 1,
  "pod_status": "Running"
}
```


Type: PodNet


```
{
  "AutoScalingGroupName": "eksctl-myCICluster-nodegroup-standard-workers-
NodeGroup-1174PV2WHZAYU",
  "ClusterName": "myCICluster",
  "InstanceId": "i-1234567890123456",
  "InstanceType": "t3.xlarge",
  "Namespace": "amazon-cloudwatch",
  "NodeName": "ip-192-0-2-0.us-west-2.compute.internal",
  "PodName": "cloudwatch-agent-statsd",
  "Service": "cloudwatch-agent-statsd",
  "Sources": [
    "cadvisor",
    "calculated"
  ],
  "Timestamp": "1567097351092",
  "Type": "PodNet",
  "Version": "0",
  "interface": "eth0",
  "kubernetes": {
    "host": "ip-192-168-75-26.us-west-2.compute.internal",
    "labels": {
      "app": "cloudwatch-agent-statsd",
      "pod-template-hash": "df44f855f"
    },
    "namespace_name": "amazon-cloudwatch",
    "pod_id": "2f4ff5ac-c813-11e9-a31d-06e9dde32928",
    "pod_name": "cloudwatch-agent-statsd-df44f855f-ts4q2",
    "pod_owners": [
```

```
      {
        "owner_kind": "Deployment",
        "owner_name": "cloudwatch-agent-statsd"
      }
    ],
    "service_name": "cloudwatch-agent-statsd"
  },
  "pod_interface_network_rx_bytes": 9880.697124714186,
  "pod_interface_network_rx_dropped": 0,
  "pod_interface_network_rx_errors": 0,
  "pod_interface_network_rx_packets": 107.80005532263283,
  "pod_interface_network_total_bytes": 10158.829201483635,
  "pod_interface_network_tx_bytes": 278.13207676944796,
  "pod_interface_network_tx_dropped": 0,
  "pod_interface_network_tx_errors": 0,
  "pod_interface_network_tx_packets": 1.146027574644318
}
```


Type: Container


```
{
  "AutoScalingGroupName": "eksctl-myCICluster-nodegroup-standard-workers-NodeGroup-
sample",
  "ClusterName": "myCICluster",
  "InstanceId": "i-1234567890123456",
  "InstanceType": "t3.xlarge",
  "Namespace": "amazon-cloudwatch",
  "NodeName": "ip-192-0-2-0.us-west-2.compute.internal",
  "PodName": "cloudwatch-agent-statsd",
  "Service": "cloudwatch-agent-statsd",
  "Sources": [
    "cadvisor",
    "pod",
    "calculated"
  ],
  "Timestamp": "1567097399912",
  "Type": "Container",
  "Version": "0",
  "container_cpu_limit": 200,
  "container_cpu_request": 200,
  "container_cpu_usage_system": 1.87958283771964,
  "container_cpu_usage_total": 6.159993652997942,
  "container_cpu_usage_user": 1.6707403001952357,
```

```
  "container_cpu_utilization": 0.15399984132494854,
  "container_memory_cache": 8192,
  "container_memory_failcnt": 0,
  "container_memory_hierarchical_pgfault": 0,
  "container_memory_hierarchical_pgmajfault": 0,
  "container_memory_limit": 104857600,
  "container_memory_mapped_file": 0,
  "container_memory_max_usage": 24580096,
  "container_memory_pgfault": 0,
  "container_memory_pgmajfault": 0,
  "container_memory_request": 104857600,
  "container_memory_rss": 22736896,
  "container_memory_swap": 0,
  "container_memory_usage": 24453120,
  "container_memory_utilization": 0.10574541028701798,
  "container_memory_working_set": 17580032,
  "container_status": "Running",
  "kubernetes": {
    "container_name": "cloudwatch-agent",
    "docker": {
      "container_id":
 "8967b6b37da239dfad197c9fdea3e5dfd35a8a759ec86e2e4c3f7b401e232706"
    },
    "host": "ip-192-168-75-26.us-west-2.compute.internal",
    "labels": {
      "app": "cloudwatch-agent-statsd",
      "pod-template-hash": "df44f855f"
    },
    "namespace_name": "amazon-cloudwatch",
    "pod_id": "2f4ff5ac-c813-11e9-a31d-06e9dde32928",
    "pod_name": "cloudwatch-agent-statsd-df44f855f-ts4q2",
    "pod_owners": [
      {
        "owner_kind": "Deployment",
        "owner_name": "cloudwatch-agent-statsd"
      }
    ],
    "service_name": "cloudwatch-agent-statsd"
  },
  "number_of_container_restarts": 0
}
```


Type: ContainerFS

```
{
  "AutoScalingGroupName": "eksctl-myCICluster-nodegroup-standard-workers-
NodeGroup-1174PV2WHZAYU",
  "ClusterName": "myCICluster",
  "EBSVolumeId": "aws://us-west-2b/vol-0a53108976d4a2fda",
  "InstanceId": "i-1234567890123456",
  "InstanceType": "t3.xlarge",
  "Namespace": "amazon-cloudwatch",
  "NodeName": "ip-192-0-2-0.us-west-2.compute.internal",
  "PodName": "cloudwatch-agent-statsd",
  "Service": "cloudwatch-agent-statsd",
  "Sources": [
    "cadvisor",
    "calculated"
  ],
  "Timestamp": "1567097399912",
  "Type": "ContainerFS",
  "Version": "0",

  "device": "/dev/nvme0n1p1",
  "fstype": "vfs",
  "kubernetes": {
    "container_name": "cloudwatch-agent",
    "docker": {
      "container_id":
 "8967b6b37da239dfad197c9fdea3e5dfd35a8a759ec86e2e4c3f7b401e232706"
    },
    "host": "ip-192-168-75-26.us-west-2.compute.internal",
    "labels": {
      "app": "cloudwatch-agent-statsd",
      "pod-template-hash": "df44f855f"
    },
    "namespace_name": "amazon-cloudwatch",
    "pod_id": "2f4ff5ac-c813-11e9-a31d-06e9dde32928",
    "pod_name": "cloudwatch-agent-statsd-df44f855f-ts4q2",
    "pod_owners": [
      {
        "owner_kind": "Deployment",
        "owner_name": "cloudwatch-agent-statsd"
      }
    ],
    "service_name": "cloudwatch-agent-statsd"
  }
```

```
}
```

Type: Cluster

```
{
  "CloudWatchMetrics": [
    {
      "Metrics": [
        {
          "Unit": "Count",
          "Name": "cluster_node_count"
        },
        {
          "Unit": "Count",
          "Name": "cluster_failed_node_count"
        }
      ],
      "Dimensions": [
        [
          "ClusterName"
        ]
      ],
      "Namespace": "ContainerInsights"
    }
  ],
  "ClusterName": "myCICluster",
  "Sources": [
    "apiserver"
  ],
  "Timestamp": "1567097534160",
  "Type": "Cluster",
  "Version": "0",
  "cluster_failed_node_count": 0,
  "cluster_node_count": 3
}
```

Type: ClusterService

```
{
  "CloudWatchMetrics": [
    {
      "Metrics": [
        {
```

```
          "Unit": "Count",
          "Name": "service_number_of_running_pods"
        }
      ],
      "Dimensions": [
        [
          "Service",
          "Namespace",
          "ClusterName"
        ],
        [
          "ClusterName"
        ]
      ],
      "Namespace": "ContainerInsights"
    }
  ],
  "ClusterName": "myCICluster",
  "Namespace": "amazon-cloudwatch",
  "Service": "cloudwatch-agent-statsd",
  "Sources": [
    "apiserver"
  ],
  "Timestamp": "1567097534160",
  "Type": "ClusterService",
  "Version": "0",
  "kubernetes": {
    "namespace_name": "amazon-cloudwatch",
    "service_name": "cloudwatch-agent-statsd"
  },
  "service_number_of_running_pods": 1
}
```


Type: ClusterNamespace


```
{
  "CloudWatchMetrics": [
    {
      "Metrics": [
        {
          "Unit": "Count",
          "Name": "namespace_number_of_running_pods"
        }
```

```
      ],
      "Dimensions": [
        [
          "Namespace",
          "ClusterName"
        ],
        [
          "ClusterName"
        ]
      ],
      "Namespace": "ContainerInsights"
    }
  ],
  "ClusterName": "myCICluster",
  "Namespace": "amazon-cloudwatch",
  "Sources": [
    "apiserver"
  ],
  "Timestamp": "1567097594160",
  "Type": "ClusterNamespace",
  "Version": "0",
  "kubernetes": {
    "namespace_name": "amazon-cloudwatch"
  },
  "namespace_number_of_running_pods": 7
}
```


Relevant fields in performance log events for Amazon EKS and Kubernetes

For Amazon EKS and Kubernetes, the containerized CloudWatch agent emits data as performance log events. This enables CloudWatch to ingest and store high-cardinality data. CloudWatch uses the data in the performance log events to create aggregated CloudWatch metrics at the cluster, node, and pod levels without the need to lose granular details.

The following table lists the fields in these performance log events that are relevant to the collection of Container Insights metric data. You can use CloudWatch Logs Insights to query for any of these fields to collect data or investigate issues. For more information, see Analyze Log Data With CloudWatch Logs Insights.

Type	Log field	Source	Formula or notes
Pod	`pod_cpu_utilization`	Calculated	Formula: `pod_cpu_usage_total / node_cpu_limit`
Pod	`pod_cpu_usage_total` `pod_cpu_usage_total` is reported in millicores.	cadvisor	
Pod	`pod_cpu_limit`	Calculated	Formula: `sum(container_cpu_limit)` `sum(container_cpu_limit)` includes already-completed pods. If any containers in the pod don't have a CPU limit defined, this field doesn't appear in the log event. This

Type	Log field	Source	Formula or notes
			includes [init containers]().
Pod	`pod_cpu_request`	Calculated	Formula: `sum(container_cpu_request)` `container_cpu_request` isn't guaranteed to be set. Only the ones that are set are included in the sum.
Pod	`pod_cpu_utilization_over_pod_limit`	Calculated	Formula: `pod_cpu_usage_total / pod_cpu_limit`
Pod	`pod_cpu_reserved_capacity`	Calculated	Formula: `pod_cpu_request / node_cpu_limit`

Type	Log field	Source	Formula or notes
Pod	`pod_memory_utilization`	Calculated	Formula: `pod_memory_working_set / node_memory_limit` It is the percentage of pod memory usage over the node memory limitation.
Pod	`pod_memory_working_set`	cadvisor	
Pod	`pod_memory_limit`	Calculated	Formula: `sum(container_memory_limit)` If any containers in the pod don't have a memory limit defined, this field doesn't appear in the log event. This includes [init containers](https://kubernetes.io/docs/concepts/workloads/pods/init-containers/).

Type	Log field	Source	Formula or notes
Pod	`pod_memory_request`	Calculated	Formula: `sum(container_memory_request)` `container_memory_request` isn't guaranteed to be set. Only the ones that are set are included in the sum.
Pod	`pod_memory_utilization_over_pod_limit`	Calculated	Formula: `pod_memory_working_set / pod_memory_limit` If any containers in the pod don't have a memory limit defined, this field doesn't appear in the log event. This includes init containers.

Type	Log field	Source	Formula or notes
Pod	pod_memory_reserved_capacity	Calculated	Formula: pod_memory_request / node_memory_limit
Pod	pod_network_tx_bytes	Calculated	Formula: sum(pod_interface_network_tx_bytes) This data is available for all the network interfaces per pod. The CloudWatch agent calculates the total and adds metric extraction rules.
Pod	pod_network_rx_bytes	Calculated	Formula: sum(pod_interface_network_rx_bytes)

Type	Log field	Source	Formula or notes
Pod	pod_network_total_bytes	Calculated	Formula: pod_network_rx_bytes + pod_network_tx_bytes
PodNet	pod_interface_network_rx_bytes	cadvisor	This data is network rx bytes per second of a pod network interface.
PodNet	pod_interface_network_tx_bytes	cadvisor	This data is network tx bytes per second of a pod network interface.
Container	container_cpu_usage_total	cadvisor	
Container	container_cpu_limit	cadvisor	Not guaranteed to be set. It's not emitted if it's not set.
Container	container_cpu_request	cadvisor	Not guaranteed to be set. It's not emitted if it's not set.
Container	container_memory_working_set	cadvisor	

Type	Log field	Source	Formula or notes
Container	`container_memory_limit`	pod	Not guaranteed to be set. It's not emitted if it's not set.
Container	`container_memory_request`	pod	Not guaranteed to be set. It's not emitted if it's not set.
Node	`node_cpu_utilization`	Calculated	Formula: `node_cpu_usage_total / node_cpu_limit`
Node	`node_cpu_usage_total`	cadvisor	
Node	`node_cpu_limit`	/proc	

Type	Log field	Source	Formula or notes
Node	`node_cpu_request`	Calculated	Formula: `sum(pod_cpu_request)` For cronjobs, `node_cpu_request` also includes requests from completed pods. This can lead to a high value for `node_cpu_reserved_capacity` .
Node	`node_cpu_reserved_capacity`	Calculated	Formula: `node_cpu_request / node_cpu_limit`
Node	`node_memory_utilization`	Calculated	Formula: `node_memory_working_set / node_memory_limit`
Node	`node_memory_working_set`	cadvisor	
Node	`node_memory_limit`	/proc	

Type	Log field	Source	Formula or notes
Node	`node_memory_request`	Calculated	Formula: `sum(pod_memory_request)`
Node	`node_memory_reserved_capacity`	Calculated	Formula: `node_memory_request / node_memory_limit`
Node	`node_network_rx_bytes`	Calculated	Formula: `sum(node_interface_network_rx_bytes)`
Node	`node_network_tx_bytes`	Calculated	Formula: `sum(node_interface_network_tx_bytes)`
Node	`node_network_total_bytes`	Calculated	Formula: `node_network_rx_bytes + node_network_tx_bytes`
Node	`node_number_of_running_pods`	Pod List	

Type	Log field	Source	Formula or notes
Node	node_number_of_running_containers	Pod List	
NodeNet	node_interface_network_rx_bytes	cadvisor	This data is network rx bytes per second of a worker node network interface.
NodeNet	node_interface_network_tx_bytes	cadvisor	This data is network tx bytes per second of a worker node network interface.
NodeFS	node_filesystem_capacity	cadvisor	
NodeFS	node_filesystem_usage	cadvisor	
NodeFS	node_filesystem_utilization	Calculated	Formula: node_filesystem_usage / node_filesystem_capacity This data is available per device name.

Type	Log field	Source	Formula or notes
Cluster	`cluster_failed_node_count`	API Server	
Cluster	`cluster_node_count`	API Server	
Service	`service_number_of_running_pods`	API Server	
Namespace	`namespace_number_of_running_pods`	API Server	

Metrics calculation examples

This section includes examples that show how some of the values in the preceding table are calculated.

Suppose that you have a cluster in the following state.

```
Node1
   node_cpu_limit = 4
   node_cpu_usage_total = 3

   Pod1
     pod_cpu_usage_total = 2

     Container1
        container_cpu_limit = 1
        container_cpu_request = 1
        container_cpu_usage_total = 0.8

     Container2
        container_cpu_limit = null
        container_cpu_request = null
        container_cpu_usage_total = 1.2

   Pod2
     pod_cpu_usage_total = 0.4

     Container3
```

```
        container_cpu_limit = 1
        container_cpu_request = 0.5
        container_cpu_usage_total = 0.4

Node2
   node_cpu_limit = 8
   node_cpu_usage_total = 1.5

   Pod3
     pod_cpu_usage_total = 1

     Container4
        container_cpu_limit = 2
        container_cpu_request = 2
        container_cpu_usage_total = 1
```

The following table shows how pod CPU metrics are calculated using this data.

Metric	Formula	Pod1	Pod2	Pod3
`pod_cpu_utilization`	`pod_cpu_usage_total / node_cpu_limit`	2 / 4 = 50%	0.4 / 4 = 10%	1 / 8 = 12.5%
`pod_cpu_utilization_over_pod_limit`	`pod_cpu_usage_total / sum(container_cpu_limit)`	N/A because CPU limit for `Container2` isn't defined	0.4 / 1 = 40%	1 / 2 = 50%
`pod_cpu_reserved_capacity`	`sum(container_cpu_request) / node_cpu_limit`	(1 + 0) / 4 = 25%	0.5 / 4 = 12.5%	2 / 8 = 25%

The following table shows how node CPU metrics are calculated using this data.

Metric	Formula	Node1	Node2
node_cpu_utilization	node_cpu_usage_total / node_cpu_limit	3 / 4 = 75%	1.5 / 8 = 18.75%
node_cpu_reserved_capacity	sum(pod_cpu_request) / node_cpu_limit	1.5 / 4 = 37.5%	2 / 8 = 25%

Container Insights Prometheus metrics monitoring

CloudWatch Container Insights monitoring for Prometheus automates the discovery of Prometheus metrics from containerized systems and workloads. Prometheus is an open-source systems monitoring and alerting toolkit. For more information, see What is Prometheus? in the Prometheus documentation.

Discovering Prometheus metrics is supported for Amazon Elastic Container Service, Amazon Elastic Kubernetes Service and Kubernetes clusters running on Amazon EC2 instances. The Prometheus counter, gauge, and summary metric types are collected.

For Amazon ECS and Amazon EKS clusters, both the EC2 and Fargate launch types are supported. Container Insights automatically collects metrics from several workloads, and you can configure it to collect metrics from any workload.

You can adopt Prometheus as an open-source and open-standard method to ingest custom metrics in CloudWatch. The CloudWatch agent with Prometheus support discovers and collects Prometheus metrics to monitor, troubleshoot, and alarm on application performance degradation and failures faster. This also reduces the number of monitoring tools required to improve observability.

Container Insights Prometheus support involves pay-per-use of metrics and logs, including collecting, storing, and analyzing. For more information, see Amazon CloudWatch Pricing.

Pre-built dashboards for some workloads

The Container Insights Prometheus solution includes pre-built dashboards for the popular workloads that are listed in this section. For sample configurations for these workloads, see (Optional) Set up sample containerized Amazon ECS workloads for Prometheus metric testing and (Optional) Set up sample containerized Amazon EKS workloads for Prometheus metric testing.

You can also configure Container Insights to collect Prometheus metrics from other containerized services and applications by editing the agent configuration file.

Workloads with pre-built dashboards for Amazon EKS clusters and Kubernetes clusters running on Amazon EC2 instances:

- Amazon App Mesh
- NGINX
- Memcached
- Java/JMX
- HAProxy

Workloads with pre-built dashboards for Amazon ECS clusters:

- Amazon App Mesh
- Java/JMX
- NGINX
- NGINX Plus

Set up and configure Prometheus metrics collection on Amazon ECS clusters

To collect Prometheus metrics from Amazon ECS clusters, you can use the CloudWatch agent as a collector or use the Amazon Distro for OpenTelemetry collector. For information about using the Amazon Distro for OpenTelemetry collector, see https://aws-otel.github.io/docs/getting-started/container-insights/ecs-prometheus.

The following sections explain how to use the CloudWatch agent as the collector to retrieve Prometheus metrics. You install the CloudWatch agent with Prometheus monitoring on clusters running Amazon ECS, and you can optionally configure the agent to scrape additional targets. These sections also provide optional tutorials for setting up sample workloads to use for testing with Prometheus monitoring.

Container Insights on Amazon ECS supports the following launch type and network mode combinations for Prometheus metrics:

Amazon ECS launch type	Network modes supported
EC2 (Linux)	bridge, host, and awsvpc
Fargate	awsvpc

VPC security group requirements

The ingress rules of the security groups for the Prometheus workloads must open the Prometheus ports to the CloudWatch agent for scraping the Prometheus metrics by the private IP.

The egress rules of the security group for the CloudWatch agent must allow the CloudWatch agent to connect to the Prometheus workloads' port by private IP.

Topics

- Install the CloudWatch agent with Prometheus metrics collection on Amazon ECS clusters
- Scraping additional Prometheus sources and importing those metrics
- (Optional) Set up sample containerized Amazon ECS workloads for Prometheus metric testing

Install the CloudWatch agent with Prometheus metrics collection on Amazon ECS clusters

This section explains how to set up the CloudWatch agent with Prometheus monitoring in a cluster running Amazon ECS. After you do this, the agent automatically scrapes and imports metrics for the following workloads running in that cluster.

- Amazon App Mesh
- Java/JMX

You can also configure the agent to scrape and import metrics from additional Prometheus workloads and sources.

Set up IAM roles

You need two IAM roles for the CloudWatch agent task definition. If you specify **`CreateIAMRoles=True`** in the Amazon CloudFormation stack to have Container Insights create these roles for you, the roles will be created with the correct permissions. If you want to create them yourself or use existing roles, the following roles and permissions are required.

- **CloudWatch agent ECS task role**— The CloudWatch agent container uses this role. It must include the **CloudWatchAgentServerPolicy** policy and a customer-managed policy which contains the following read-only permissions:
  - `ec2:DescribeInstances`
  - `ecs:ListTasks`
  - `ecs:ListServices`
  - `ecs:DescribeContainerInstances`
  - `ecs:DescribeServices`
  - `ecs:DescribeTasks`
  - `ecs:DescribeTaskDefinition`
- **CloudWatch agent ECS task execution role**— This is the role that Amazon ECS requires to launch and execute your containers. Ensure that your task execution role has the **AmazonSSMReadOnlyAccess**, **AmazonECSTaskExecutionRolePolicy**, and **CloudWatchAgentServerPolicy** policies attached. If you want to store more sensitive data for Amazon ECS to use, see [Specifying sensitive data](#).

Install the CloudWatch agent with Prometheus monitoring by using Amazon CloudFormation

You use Amazon CloudFormation to install the CloudWatch agent with Prometheus monitoring for Amazon ECS clusters. The following list shows the parameters you will use in the Amazon CloudFormation template.

- **ECSClusterName**— Specifies the target Amazon ECS cluster.
- **CreateIAMRoles**— Specify **`True`** to create new roles for the Amazon ECS task role and Amazon ECS task execution role. Specify **`False`** to reuse existing roles.
- **TaskRoleName**— If you specified **`True`** for **CreateIAMRoles**, this specifies the name to use for the new Amazon ECS task role. If you specified **`False`** for **CreateIAMRoles**, this specifies the existing role to use as the Amazon ECS task role.
- **ExecutionRoleName**— If you specified **`True`** for **CreateIAMRoles**, this specifies the name to use for the new Amazon ECS task execution role. If you specified **`False`** for **CreateIAMRoles**, this specifies the existing role to use as the Amazon ECS task execution role.
- **ECSNetworkMode**— If you are using EC2 launch type, specify the network mode here. It must be either **`bridge`** or **`host`**.
- **ECSLaunchType**— Specify either **`fargate`** or **EC2**.

- **SecurityGroupID**— If the **ECSNetworkMode** is **awsvpc**, specify the security group ID here.
- **SubnetID**— If the **ECSNetworkMode** is **awsvpc**, specify the subnet ID here.

Command samples

This section includes sample Amazon CloudFormation commands to install Container Insights with Prometheus monitoring in various scenarios.

Create Amazon CloudFormation stack for an Amazon ECS cluster in bridge network mode

```
export AWS_PROFILE=your_aws_config_profile_eg_default
export AWS_DEFAULT_REGION=your_aws_region_eg_ap-southeast-1
export ECS_CLUSTER_NAME=your_ec2_ecs_cluster_name
export ECS_NETWORK_MODE=bridge
export CREATE_IAM_ROLES=True
export ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
export ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name

curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
insights/latest/ecs-task-definition-templates/deployment-mode/replica-service/cwagent-
prometheus/cloudformation-quickstart/cwagent-ecs-prometheus-metric-for-bridge-host.yaml

aws cloudformation create-stack --stack-name CWAgent-Prometheus-ECS-
${ECS_CLUSTER_NAME}-EC2-${ECS_NETWORK_MODE} \
    --template-body file://cwagent-ecs-prometheus-metric-for-bridge-host.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=${ECS_CLUSTER_NAME} \
                 ParameterKey=CreateIAMRoles,ParameterValue=${CREATE_IAM_ROLES} \
                 ParameterKey=ECSNetworkMode,ParameterValue=${ECS_NETWORK_MODE} \
                 ParameterKey=TaskRoleName,ParameterValue=${ECS_TASK_ROLE_NAME} \
                 ParameterKey=ExecutionRoleName,ParameterValue=
${ECS_EXECUTION_ROLE_NAME} \
    --capabilities CAPABILITY_NAMED_IAM \
    --region ${AWS_DEFAULT_REGION} \
    --profile ${AWS_PROFILE}
```

Create Amazon CloudFormation stack for an Amazon ECS cluster in host network mode

```
export AWS_PROFILE=your_aws_config_profile_eg_default
export AWS_DEFAULT_REGION=your_aws_region_eg_ap-southeast-1
export ECS_CLUSTER_NAME=your_ec2_ecs_cluster_name
export ECS_NETWORK_MODE=host
export CREATE_IAM_ROLES=True
```

```
export ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
export ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name


curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
insights/latest/ecs-task-definition-templates/deployment-mode/replica-service/cwagent-
prometheus/cloudformation-quickstart/cwagent-ecs-prometheus-metric-for-bridge-host.yaml

aws cloudformation create-stack --stack-name CWAgent-Prometheus-ECS-
${ECS_CLUSTER_NAME}-EC2-${ECS_NETWORK_MODE} \
    --template-body file://cwagent-ecs-prometheus-metric-for-bridge-host.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=${ECS_CLUSTER_NAME} \
                 ParameterKey=CreateIAMRoles,ParameterValue=${CREATE_IAM_ROLES} \
                 ParameterKey=ECSNetworkMode,ParameterValue=${ECS_NETWORK_MODE} \
                 ParameterKey=TaskRoleName,ParameterValue=${ECS_TASK_ROLE_NAME} \
                 ParameterKey=ExecutionRoleName,ParameterValue=
${ECS_EXECUTION_ROLE_NAME} \
    --capabilities CAPABILITY_NAMED_IAM \
    --region ${AWS_DEFAULT_REGION} \
    --profile ${AWS_PROFILE}
```

Create Amazon CloudFormation stack for an Amazon ECS cluster in awsvpc network mode

```
export AWS_PROFILE=your_aws_config_profile_eg_default
export AWS_DEFAULT_REGION=your_aws_region_eg_ap-southeast-1
export ECS_CLUSTER_NAME=your_ec2_ecs_cluster_name
export ECS_LAUNCH_TYPE=EC2
export CREATE_IAM_ROLES=True
export ECS_CLUSTER_SECURITY_GROUP=your_security_group_eg_sg-xxxxxxxxxx
export ECS_CLUSTER_SUBNET=your_subnet_eg_subnet-xxxxxxxxxx
export ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
export ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name

curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
insights/latest/ecs-task-definition-templates/deployment-mode/replica-service/cwagent-
prometheus/cloudformation-quickstart/cwagent-ecs-prometheus-metric-for-awsvpc.yaml

aws cloudformation create-stack --stack-name CWAgent-Prometheus-ECS-
${ECS_CLUSTER_NAME}-${ECS_LAUNCH_TYPE}-awsvpc \
    --template-body file://cwagent-ecs-prometheus-metric-for-awsvpc.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=${ECS_CLUSTER_NAME} \
                 ParameterKey=CreateIAMRoles,ParameterValue=${CREATE_IAM_ROLES} \
                 ParameterKey=ECSLaunchType,ParameterValue=${ECS_LAUNCH_TYPE} \
```

```
                 ParameterKey=SecurityGroupID,ParameterValue=
${ECS_CLUSTER_SECURITY_GROUP} \
                 ParameterKey=SubnetID,ParameterValue=${ECS_CLUSTER_SUBNET} \
                 ParameterKey=TaskRoleName,ParameterValue=${ECS_TASK_ROLE_NAME} \
                 ParameterKey=ExecutionRoleName,ParameterValue=
${ECS_EXECUTION_ROLE_NAME} \
    --capabilities CAPABILITY_NAMED_IAM \
    --region ${AWS_DEFAULT_REGION} \
    --profile ${AWS_PROFILE}
```

Create Amazon CloudFormation stack for a Fargate cluster in awsvpc network mode

```
export AWS_PROFILE=your_aws_config_profile_eg_default
export AWS_DEFAULT_REGION=your_aws_region_eg_ap-southeast-1
export ECS_CLUSTER_NAME=your_ec2_ecs_cluster_name
export ECS_LAUNCH_TYPE=FARGATE
export CREATE_IAM_ROLES=True
export ECS_CLUSTER_SECURITY_GROUP=your_security_group_eg_sg-xxxxxxxxxx
export ECS_CLUSTER_SUBNET=your_subnet_eg_subnet-xxxxxxxxxx
export ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
export ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name

curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
insights/latest/ecs-task-definition-templates/deployment-mode/replica-service/cwagent-
prometheus/cloudformation-quickstart/cwagent-ecs-prometheus-metric-for-awsvpc.yaml

aws cloudformation create-stack --stack-name CWAgent-Prometheus-ECS-
${ECS_CLUSTER_NAME}-${ECS_LAUNCH_TYPE}-awsvpc \
    --template-body file://cwagent-ecs-prometheus-metric-for-awsvpc.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=${ECS_CLUSTER_NAME} \
                 ParameterKey=CreateIAMRoles,ParameterValue=${CREATE_IAM_ROLES} \
                 ParameterKey=ECSLaunchType,ParameterValue=${ECS_LAUNCH_TYPE} \
                 ParameterKey=SecurityGroupID,ParameterValue=
${ECS_CLUSTER_SECURITY_GROUP} \
                 ParameterKey=SubnetID,ParameterValue=${ECS_CLUSTER_SUBNET} \
                 ParameterKey=TaskRoleName,ParameterValue=${ECS_TASK_ROLE_NAME} \
                 ParameterKey=ExecutionRoleName,ParameterValue=
${ECS_EXECUTION_ROLE_NAME} \
    --capabilities CAPABILITY_NAMED_IAM \
    --region ${AWS_DEFAULT_REGION} \
    --profile ${AWS_PROFILE}
```

Amazon resources created by the Amazon CloudFormation stack

The following table lists the Amazon resources that are created when you use Amazon CloudFormation to set up Container Insights with Prometheus monitoring on an Amazon ECS cluster.

Resource type	Resource name	Comments
AWS::SSM::Parameter	AmazonCloudWatch-CWAgentConfig-$*ECS_CLUSTER_NAME* -$*ECS_LAUNCH_TYPE* -$*ECS_NETWORK_MODE*	This is the CloudWatch agent with the default App Mesh and Java/JMX embedded metric format definition.
AWS::SSM::Parameter	AmazonCloudWatch-PrometheusConfigName-$*ECS_CLUSTER_NAME* -$*ECS_LAUNCH_TYPE* -$*ECS_NETWORK_MODE*	This is the Prometheus scraping configuration.
AWS::IAM::Role	**$ECS_TASK_ROLE_NAME**.	The Amazon ECS task role. This is created only if you specified **`True`** for `CREATE_IAM_ROLES` .
AWS::IAM::Role	**${ECS_EXECUTION_ROLE_NAME}**	The Amazon ECS task execution role. This is created only if you specified **`True`** for `CREATE_IAM_ROLES` .
AWS::ECS::TaskDefinition	cwagent-prometheus-$*ECS_CLUSTER_NAME* -$*ECS_LAUNCH_TYPE* -$*ECS_NETWORK_MODE*	
AWS::ECS::Service	cwagent-prometheus-replica-service-$*ECS_LAUNCH_TYPE* -$*ECS_NETWORK_MODE*	

Deleting the Amazon CloudFormation stack for the CloudWatch agent with Prometheus monitoring

To delete the CloudWatch agent from an Amazon ECS cluster, enter these commands.

```
export AWS_PROFILE=your_aws_config_profile_eg_default
export AWS_DEFAULT_REGION=your_aws_region_eg_ap-southeast-1
export CLOUDFORMATION_STACK_NAME=your_cloudformation_stack_name

aws cloudformation delete-stack \
--stack-name ${CLOUDFORMATION_STACK_NAME} \
--region ${AWS_DEFAULT_REGION} \
--profile ${AWS_PROFILE}
```

Scraping additional Prometheus sources and importing those metrics

The CloudWatch agent with Prometheus monitoring needs two configurations to scrape the Prometheus metrics. One is for the standard Prometheus configurations as documented in <scrape_config> in the Prometheus documentation. The other is for the CloudWatch agent configuration.

For Amazon ECS clusters, the configurations are integrated with the Parameter Store of Amazon Systems Manager by the secrets in the Amazon ECS task definition:

- The secret `PROMETHEUS_CONFIG_CONTENT` is for the Prometheus scrape configuration.
- The secret `CW_CONFIG_CONTENT` is for the CloudWatch agent configuration.

To scrape additional Prometheus metrics sources and import those metrics to CloudWatch, you modify both the Prometheus scrape configuration and the CloudWatch agent configuration, and then re-deploy the agent with the updated configuration.

VPC security group requirements

The ingress rules of the security groups for the Prometheus workloads must open the Prometheus ports to the CloudWatch agent for scraping the Prometheus metrics by the private IP.

The egress rules of the security group for the CloudWatch agent must allow the CloudWatch agent to connect to the Prometheus workloads' port by private IP.

Prometheus scrape configuration

The CloudWatch agent supports the standard Prometheus scrape configurations as documented in <scrape_config> in the Prometheus documentation. You can edit this section to update the configurations that are already in this file, and add additional Prometheus scraping targets. By default, the sample configuration file contains the following global configuration lines:

```
global:
  scrape_interval: 1m
  scrape_timeout: 10s
```

- **scrape_interval**— Defines how frequently to scrape targets.
- **scrape_timeout**— Defines how long to wait before a scrape request times out.

You can also define different values for these settings at the job level, to override the global configurations.

Prometheus scraping jobs

The CloudWatch agent YAML files already have some default scraping jobs configured. For example, in the YAML files for Amazon ECS such as `cwagent-ecs-prometheus-metric-for-bridge-host.yaml`, the default scraping jobs are configured in the `ecs_service_discovery` section.

```
"ecs_service_discovery": {
                  "sd_frequency": "1m",
                  "sd_result_file": "/tmp/cwagent_ecs_auto_sd.yaml",
                  "docker_label": {
                  },
                  "task_definition_list": [
                    {
                      "sd_job_name": "ecs-appmesh-colors",
                      "sd_metrics_ports": "9901",
                      "sd_task_definition_arn_pattern": ".*:task-definition\/.*-
ColorTeller-(white):[0-9]+",
                      "sd_metrics_path": "/stats/prometheus"
                    },
                    {
                      "sd_job_name": "ecs-appmesh-gateway",
                      "sd_metrics_ports": "9901",
                      "sd_task_definition_arn_pattern": ".*:task-definition/.*-
ColorGateway:[0-9]+",
                      "sd_metrics_path": "/stats/prometheus"
                    }
                  ]
                }
```

Each of these default targets are scraped, and the metrics are sent to CloudWatch in log events using embedded metric format. For more information, see [Embedding metrics within logs](#).

Log events from Amazon ECS clusters are stored in the **/aws/ecs/containerinsights/*`cluster_name`*/prometheus** log group.

Each scraping job is contained in a different log stream in this log group.

To add a new scraping target, you add a new entry in the `task_definition_list` section under the `ecs_service_discovery` section. of the YAML file, and restart the agent. For an example of this process, see [Tutorial for adding a new Prometheus scrape target: Prometheus API Server metrics](#).

CloudWatch agent configuration for Prometheus

The CloudWatch agent configuration file has a `prometheus` section under `metrics_collected` for the Prometheus scraping configuration. It includes the following configuration options:

- **cluster_name**— specifies the cluster name to be added as a label in the log event. This field is optional. If you omit it, the agent can detect the Amazon ECS cluster name.
- **log_group_name**— specifies the log group name for the scraped Prometheus metrics. This field is optional. If you omit it, CloudWatch uses **/aws/ecs/containerinsights/*`cluster_name`*/prometheus** for logs from Amazon ECS clusters.
- **prometheus_config_path**— specifies the Prometheus scrape configuration file path. If the value of this field starts with `env:` the Prometheus scrape configuration file contents will be retrieved from the container's environment variable. Do not change this field.
- **ecs_service_discovery**— is the section to specify the configurations of the Amazon ECS Prometheus target auto-discovery functions. Two modes are supported to discover the Prometheus targets: discovery based on the container's docker label or discovery based on the Amazon ECS task definition ARN regular expression. You can use the two modes together and the CloudWatch agent will de-duplicate the discovered targets based on: *{private_ip}:{port}/{metrics_path}*.

  The `ecs_service_discovery` section can contain the following fields:

  - `sd_frequency` is the frequency to discover the Prometheus exporters. Specify a number and a unit suffix. For example, `1m` for once per minute or `30s` for once per 30 seconds. Valid unit suffixes are `ns`, `us`, `ms`, `s`, `m`, and `h`.

    This field is optional. The default is 60 seconds (1 minute).

- `sd_target_cluster` is the target Amazon ECS cluster name for auto-discovery. This field is optional. The default is the name of the Amazon ECS cluster where the CloudWatch agent is installed.
- `sd_cluster_region` is the target Amazon ECS cluster's Region. This field is optional. The default is the Region of the Amazon ECS cluster where the CloudWatch agent is installed. .
- `sd_result_file` is the path of the YAML file for the Prometheus target results. The Prometheus scrape configuration will refer to this file.
- `docker_label` is an optional section that you can use to specify the configuration for docker label-based service discovery. If you omit this section, docker label-based discovery is not used. This section can contain the following fields:
  - `sd_port_label` is the container's docker label name that specifies the container port for Prometheus metrics. The default value is `ECS_PROMETHEUS_EXPORTER_PORT`. If the container does not have this docker label, the CloudWatch agent will skip it.
  - `sd_metrics_path_label` is the container's docker label name that specifies the Prometheus metrics path. The default value is `ECS_PROMETHEUS_METRICS_PATH`. If the container does not have this docker label, the agent assumes the default path `/metrics`.
  - `sd_job_name_label` is the container's docker label name that specifies the Prometheus scrape job name. The default value is `job`. If the container does not have this docker label, the CloudWatch agent uses the job name in the Prometheus scrape configuration.
- `task_definition_list` is an optional section that you can use to specify the configuration of task definition-based service discovery. If you omit this section, task definition-based discovery is not used. This section can contain the following fields:
  - `sd_task_definition_arn_pattern` is the pattern to use to specify the Amazon ECS task definitions to discover. This is a regular expression.
  - `sd_metrics_ports` lists the containerPort for the Prometheus metrics. Separate the containerPorts with semicolons.
  - `sd_container_name_pattern` specifies the Amazon ECS task container names. This is a regular expression.
  - `sd_metrics_path` specifies the Prometheus metric path. If you omit this, the agent assumes the default path `/metrics`
  - `sd_job_name` specifies the Prometheus scrape job name. If you omit this field, the CloudWatch agent uses the job name in the Prometheus scrape configuration.

- `service_name_list_for_tasks` is an optional section that you can use to specify the configuration of service name-based discovery. If you omit this section, service name-based discovery is not used. This section can contain the following fields:
  - `sd_service_name_pattern` is the pattern to use to specify the Amazon ECS service where tasks are to be discovered. This is a regular expression.
  - `sd_metrics_ports` Lists the `containerPort` for the Prometheus metrics. Separate multiple `containerPorts` with semicolons.
  - `sd_container_name_pattern` specifies the Amazon ECS task container names. This is a regular expression.
  - `sd_metrics_path` specifies the Prometheus metrics path. If you omit this, the agent assumes that the default path `/metrics`.
  - `sd_job_name` specifies the Prometheus scrape job name. If you omit this field, the CloudWatch agent uses the job name in the Prometheus scrape configuration.
- **metric_declaration**— are sections that specify the array of logs with embedded metric format to be generated. There are `metric_declaration` sections for each Prometheus source that the CloudWatch agent imports from by default. These sections each include the following fields:
  - `label_matcher` is a regular expression that checks the value of the labels listed in `source_labels`. The metrics that match are enabled for inclusion in the embedded metric format sent to CloudWatch.

    If you have multiple labels specified in `source_labels`, we recommend that you do not use ^ or $ characters in the regular expression for `label_matcher`.
  - `source_labels` specifies the value of the labels that are checked by the `label_matcher` line.
  - `label_separator` specifies the separator to be used in the `label_matcher` line if multiple `source_labels` are specified. The default is `;`. You can see this default used in the `label_matcher` line in the following example.
  - `metric_selectors` is a regular expression that specifies the metrics to be collected and sent to CloudWatch.
  - `dimensions` is the list of labels to be used as CloudWatch dimensions for each selected metric.

See the following `metric_declaration` example.

```
"metric_declaration": [
  {
     "source_labels":[ "Service", "Namespace"],
     "label_matcher":"(.*node-exporter.*|.*kube-dns.*);kube-system$",
     "dimensions":[
        ["Service", "Namespace"]
     ],
     "metric_selectors":[
        "^coredns_dns_request_type_count_total$"
     ]
  }
]
```

This example configures an embedded metric format section to be sent as a log event if the following conditions are met:

- The value of `Service` contains either `node-exporter` or `kube-dns`.
- The value of `Namespace` is `kube-system`.
- The Prometheus metric `coredns_dns_request_type_count_total` contains both `Service` and `Namespace` labels.

The log event that is sent includes the following highlighted section:

```
{
   "CloudWatchMetrics":[
      {
         "Metrics":[
            {
               "Name":"coredns_dns_request_type_count_total"
            }
         ],
         "Dimensions":[
            [
               "Namespace",
               "Service"
            ]
         ],
         "Namespace":"ContainerInsights/Prometheus"
      }
   ],
   "Namespace":"kube-system",
```

```
    "Service":"kube-dns",
    "coredns_dns_request_type_count_total":2562,
    "eks_amazonaws_com_component":"kube-dns",
    "instance":"192.168.61.254:9153",
    "job":"kubernetes-service-endpoints",
    ...
}
```

Detailed guide for autodiscovery on Amazon ECS clusters

Prometheus provides dozens of dynamic service-discovery mechanisms as described in <scrape_config>. However there is no built-in service discovery for Amazon ECS. The CloudWatch agent adds this mechanism.

When the Amazon ECS Prometheus service discovery is enabled, the CloudWatch agent periodically makes the following API calls to Amazon ECS and Amazon EC2 frontends to retrieve the metadata of the running ECS tasks in the target ECS cluster.

```
EC2:DescribeInstances
ECS:ListTasks
ECS:ListServices
ECS:DescribeContainerInstances
ECS:DescribeServices
ECS:DescribeTasks
ECS:DescribeTaskDefinition
```

The metadata is used by the CloudWatch agent to scan the Prometheus targets within the ECS cluster. The CloudWatch agent supports three service discovery modes:

- Container docker label-based service discovery
- ECS task definition ARN regular expression-based service discovery
- ECS service name regular expression-based service discovery

All modes can be used together. CloudWatch agent de-duplicates the discovered targets based on: `{private_ip}:{port}/{metrics_path}`.

All discovered targets are written into a result file specified by the `sd_result_file` configuration field within the CloudWatch agent container. The following is a sample result file:

```
- targets:
```

```
  - 10.6.1.95:32785
  labels:
    __metrics_path__: /metrics
    ECS_PROMETHEUS_EXPORTER_PORT: "9406"
    ECS_PROMETHEUS_JOB_NAME: demo-jar-ec2-bridge-dynamic
    ECS_PROMETHEUS_METRICS_PATH: /metrics
    InstanceType: t3.medium
    LaunchType: EC2
    SubnetId: subnet-123456789012
    TaskDefinitionFamily: demo-jar-ec2-bridge-dynamic-port
    TaskGroup: family:demo-jar-ec2-bridge-dynamic-port
    TaskRevision: "7"
    VpcId: vpc-01234567890
    container_name: demo-jar-ec2-bridge-dynamic-port
    job: demo-jar-ec2-bridge-dynamic
- targets:
  - 10.6.3.193:9404
  labels:
    __metrics_path__: /metrics
    ECS_PROMETHEUS_EXPORTER_PORT_SUBSET_B: "9404"
    ECS_PROMETHEUS_JOB_NAME: demo-tomcat-ec2-bridge-mapped-port
    ECS_PROMETHEUS_METRICS_PATH: /metrics
    InstanceType: t3.medium
    LaunchType: EC2
    SubnetId: subnet-123456789012
    TaskDefinitionFamily: demo-tomcat-ec2-bridge-mapped-port
    TaskGroup: family:demo-jar-tomcat-bridge-mapped-port
    TaskRevision: "12"
    VpcId: vpc-01234567890
    container_name: demo-tomcat-ec2-bridge-mapped-port
    job: demo-tomcat-ec2-bridge-mapped-port
```

You can directly integrate this result file with Prometheus file-based service discovery. For more information about Prometheus file-based service discovery, see <file_sd_config>.

Suppose the result file is written to `/tmp/cwagent_ecs_auto_sd.yaml` The following Prometheus scrape configuration will consume it.

```
global:
  scrape_interval: 1m
  scrape_timeout: 10s
scrape_configs:
  - job_name: cwagent-ecs-file-sd-config
```

```
    sample_limit: 10000
    file_sd_configs:
      - files: [ "/tmp/cwagent_ecs_auto_sd.yaml" ]
```

The CloudWatch agent also adds the following additional labels for the discovered targets.

- `container_name`
- `TaskDefinitionFamily`
- `TaskRevision`
- `TaskGroup`
- `StartedBy`
- `LaunchType`
- `job`
- `__metrics_path__`
- Docker labels

When the cluster has the EC2 launch type, the following three labels are added.

- `InstanceType`
- `VpcId`
- `SubnetId`

> **Note**
>
> Docker labels that don't match the regular expression `[a-zA-Z_][a-zA-Z0-9_]*` are filtered out. This matches the Prometheus conventions as listed in `label_name` in [Configuration file](#) in the Prometheus documentation.

ECS service discovery configuration examples

This section includes examples that demonstrate ECS service discovery.

Example 1

```
"ecs_service_discovery": {
```

```
  "sd_frequency": "1m",
  "sd_result_file": "/tmp/cwagent_ecs_auto_sd.yaml",
  "docker_label": {
  }
}
```

This example enables docker label-based service discovery. The CloudWatch agent will query the ECS tasks' metadata once per minute and write the discovered targets into the `/tmp/cwagent_ecs_auto_sd.yaml` file within the CloudWatch agent container.

The default value of `sd_port_label` in the `docker_label` section is `ECS_PROMETHEUS_EXPORTER_PORT`. If any running container in the ECS tasks has a `ECS_PROMETHEUS_EXPORTER_PORT` docker label, the CloudWatch agent uses its value as `container port` to scan all exposed ports of the container. If there is a match, the mapped host port plus the private IP of the container are used to construct the Prometheus exporter target in the following format: `private_ip:host_port`.

The default value of `sd_metrics_path_label` in the `docker_label` section is `ECS_PROMETHEUS_METRICS_PATH`. If the container has this docker label, its value will be used as the `__metrics_path__` . If the container does not have this label, the default value `/metrics` is used.

The default value of `sd_job_name_label` in the `docker_label` section is `job`. If the container has this docker label, its value will be appended as one of the labels for the target to replace the default job name specified in the Prometheus configuration. The value of this docker label is used as the log stream name in the CloudWatch Logs log group.

Example 2

```
"ecs_service_discovery": {
  "sd_frequency": "15s",
  "sd_result_file": "/tmp/cwagent_ecs_auto_sd.yaml",
  "docker_label": {
    "sd_port_label": "ECS_PROMETHEUS_EXPORTER_PORT_SUBSET_A",
    "sd_job_name_label": "ECS_PROMETHEUS_JOB_NAME"
  }
}
```

This example enables docker label-based service discovery. THe CloudWatch agent will query the ECS tasks' metadata every 15 seconds and write the discovered targets into the `/tmp/`

`cwagent_ecs_auto_sd.yaml` file within the CloudWatch agent container. The containers with a docker label of `ECS_PROMETHEUS_EXPORTER_PORT_SUBSET_A` will be scanned. The value of the docker label `ECS_PROMETHEUS_JOB_NAME` is used as the job name.

Example 3

```
"ecs_service_discovery": {
  "sd_frequency": "5m",
  "sd_result_file": "/tmp/cwagent_ecs_auto_sd.yaml",
  "task_definition_list": [
    {
      "sd_job_name": "java-prometheus",
      "sd_metrics_path": "/metrics",
      "sd_metrics_ports": "9404; 9406",
      "sd_task_definition_arn_pattern": ".*:task-definition/.*javajmx.*:[0-9]+"
    },
    {
      "sd_job_name": "envoy-prometheus",
      "sd_metrics_path": "/stats/prometheus",
      "sd_container_name_pattern": "^envoy$",
      "sd_metrics_ports": "9901",
      "sd_task_definition_arn_pattern": ".*:task-definition/.*appmesh.*:23"
    }
  ]
}
```

This example enables ECS task definition ARN regular expression-based service discovery. The CloudWatch agent will query the ECS tasks' metadata every five minutes and write the discovered targets into the `/tmp/cwagent_ecs_auto_sd.yaml` file within the CloudWatch agent container.

Two task definition ARN regular expresion sections are defined:

- For the first section, the ECS tasks with `javajmx` in their ECS task definition ARN are filtered for the container port scan. If the containers within these ECS tasks expose the container port on 9404 or 9406, the mapped host port along with the private IP of the container are used to create the Prometheus exporter targets. The value of `sd_metrics_path` sets `__metrics_path__` to `/metrics`. So the CloudWatch agent will scrape the Prometheus metrics from `private_ip:host_port/metrics`, the scraped metrics are sent to the `java-prometheus` log stream in CloudWatch Logs in the log group `/aws/ecs/containerinsights/cluster_name/prometheus`.

- For the second section, the ECS tasks with `appmesh` in their ECS task definition ARN and with `version` of `:23` are filtered for the container port scan. For containers with a name of `envoy` that expose the container port on `9901`, the mapped host port along with the private IP of the container are used to create the Prometheus exporter targets. The value within these ECS tasks expose the container port on 9404 or 9406, the mapped host port along with the private IP of the container are used to create the Prometheus exporter targets. The value of `sd_metrics_path` sets `__metrics_path__` to `/stats/prometheus`. So the CloudWatch agent will scrape the Prometheus metrics from `private_ip:host_port/stats/prometheus`, and send the scraped metrics to the `envoy-prometheus` log stream in CloudWatch Logs in the log group `/aws/ecs/containerinsights/cluster_name/prometheus`.

Example 4

```
"ecs_service_discovery": {
  "sd_frequency": "5m",
  "sd_result_file": "/tmp/cwagent_ecs_auto_sd.yaml",
  "service_name_list_for_tasks": [
    {
      "sd_job_name": "nginx-prometheus",
      "sd_metrics_path": "/metrics",
      "sd_metrics_ports": "9113",
      "sd_service_name_pattern": "^nginx-.*"
    },
    {
      "sd_job_name": "haproxy-prometheus",
      "sd_metrics_path": "/stats/metrics",
      "sd_container_name_pattern": "^haproxy$",
      "sd_metrics_ports": "8404",
      "sd_service_name_pattern": ".*haproxy-service.*"
    }
  ]
}
```

This example enables ECS service name regular expression-based service discovery. The CloudWatch agent will query the ECS services' metadata every five minutes and write the discovered targets into the `/tmp/cwagent_ecs_auto_sd.yaml` file within the CloudWatch agent container.

Two service name regular expresion sections are defined:

- For the first section, the ECS tasks that are associated with ECS services that have names matching the regular expression `^nginx-.*` are filtered for the container port scan. If the containers within these ECS tasks expose the container port on 9113, the mapped host port along with the private IP of the container are used to create the Prometheus exporter targets. The value of `sd_metrics_path` sets `__metrics_path__` to `/metrics`. So the CloudWatch agent will scrape the Prometheus metrics from `private_ip:host_port/metrics`, and the scraped metrics are sent to the `nginx-prometheus` log stream in CloudWatch Logs in the log group `/aws/ecs/containerinsights/cluster_name/prometheus`.
- or the second section, the ECS tasks that are associated with ECS services that have names matching the regular expression `.*haproxy-service.*` are filtered for the container port scan. For containers with a name of `haproxy` expose the container port on 8404, the mapped host port along with the private IP of the container are used to create the Prometheus exporter targets. The value of `sd_metrics_path` sets `__metrics_path__` to `/stats/metrics`. So the CloudWatch agent will scrape the Prometheus metrics from `private_ip:host_port/stats/metrics`, and the scraped metrics are sent to the `haproxy-prometheus` log stream in CloudWatch Logs in the log group `/aws/ecs/containerinsights/cluster_name/prometheus`.

Example 5

```
"ecs_service_discovery": {
  "sd_frequency": "1m30s",
  "sd_result_file": "/tmp/cwagent_ecs_auto_sd.yaml",
  "docker_label": {
    "sd_port_label": "MY_PROMETHEUS_EXPORTER_PORT_LABEL",
    "sd_metrics_path_label": "MY_PROMETHEUS_METRICS_PATH_LABEL",
    "sd_job_name_label": "MY_PROMETHEUS_METRICS_NAME_LABEL"
  }
  "task_definition_list": [
    {
      "sd_metrics_ports": "9150",
      "sd_task_definition_arn_pattern": "*memcached.*"
    }
  ]
}
```

This example enables both ECS service discovery modes. The CloudWatch agent will query the ECS tasks' metadata every 90 seconds and write the discovered targets into the `/tmp/cwagent_ecs_auto_sd.yaml` file within the CloudWatch agent container.

For the docker-based service discovery configuration:

- The ECS tasks with docker label `MY_PROMETHEUS_EXPORTER_PORT_LABEL` will be filtered for Prometheus port scan. The target Prometheus container port is specified by the value of the label `MY_PROMETHEUS_EXPORTER_PORT_LABEL`.
- The value of the docker label `MY_PROMETHEUS_EXPORTER_PORT_LABEL` is used for `__metrics_path__`. If the container does not have this docker label, the default value `/metrics` is used.
- The value of the docker label `MY_PROMETHEUS_EXPORTER_PORT_LABEL` is used as the job label. If the container does not have this docker label, the job name defined in the Prometheus configuration is used.

For the ECS task definition ARN regular expression-based service discovery configuration:

- The ECS tasks with `memcached` in the ECS task definition ARN are filtered for container port scan. The target Prometheus container port is 9150 as defined by `sd_metrics_ports`. The default metrics path `/metrics` is used. The job name defined in the Prometheus configuration is used.

(Optional) Set up sample containerized Amazon ECS workloads for Prometheus metric testing

To test the Prometheus metric support in CloudWatch Container Insights, you can set up one or more of the following containerized workloads. The CloudWatch agent with Prometheus support automatically collects metrics from each of these workloads. To see the metrics that are collected by default, see Prometheus metrics collected by the CloudWatch agent.

Topics

- Tutorial for scraping Redis OSS Prometheus metrics on Amazon ECS Fargate

Sample App Mesh workload for Amazon ECS clusters

To collect metrics from a sample Prometheus workload for Amazon ECS, you must be running Container Insights in the cluster. For information about installing Container Insights, see Setting up Container Insights on Amazon ECS.

First, follow this walkthrough to deploy the sample color app on your Amazon ECS cluster. After you finish, you will have App Mesh Prometheus metrics exposed on port 9901.

Next, follow these steps to install the CloudWatch agent with Prometheus monitoring on the same Amazon ECS cluster where you installed the color app. The steps in this section install the CloudWatch agent in bridge network mode.

The environment variables `ENVIRONMENT_NAME`, `AWS_PROFILE`, and `AWS_DEFAULT_REGION` that you set in the walkthrough will also be used in the following steps.

To install the CloudWatch agent with Prometheus monitoring for testing

1. Download the Amazon CloudFormation template by entering the following command.

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
insights/latest/ecs-task-definition-templates/deployment-mode/replica-service/
cwagent-prometheus/cloudformation-quickstart/cwagent-ecs-prometheus-metric-for-
bridge-host.yaml
```

2. Set the network mode by entering the following commands.

```
export ECS_CLUSTER_NAME=${ENVIRONMENT_NAME}
export ECS_NETWORK_MODE=bridge
```

3. Create the Amazon CloudFormation stack by entering the following commands.

```
aws cloudformation create-stack --stack-name CWAgent-Prometheus-ECS-
${ECS_CLUSTER_NAME}-EC2-${ECS_NETWORK_MODE} \
    --template-body file://cwagent-ecs-prometheus-metric-for-bridge-host.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=${ECS_CLUSTER_NAME} \
                 ParameterKey=CreateIAMRoles,ParameterValue=True \
                 ParameterKey=ECSNetworkMode,ParameterValue=${ECS_NETWORK_MODE} \
                 ParameterKey=TaskRoleName,ParameterValue=CWAgent-Prometheus-
TaskRole-${ECS_CLUSTER_NAME} \
```

```
                 ParameterKey=ExecutionRoleName,ParameterValue=CWAgent-Prometheus-ExecutionRole-${ECS_CLUSTER_NAME} \
    --capabilities CAPABILITY_NAMED_IAM \
    --region ${AWS_DEFAULT_REGION} \
    --profile ${AWS_PROFILE}
```

4. (Optional) When the Amazon CloudFormation stack is created, you see a `CREATE_COMPLETE` message. If you to check the status before you see that message, enter the following command.

```
aws cloudformation describe-stacks \
--stack-name CWAgent-Prometheus-ECS-${ECS_CLUSTER_NAME}-EC2-${ECS_NETWORK_MODE} \
--query 'Stacks[0].StackStatus' \
--region ${AWS_DEFAULT_REGION} \
--profile ${AWS_PROFILE}
```

Troubleshooting

The steps in the walkthrough use jq to parse the output result of the Amazon CLI. For more information about installing jq, see [jq](). Use the following command to set the default output format of your Amazon CLI to JSON so jq can parse it correctly.

```
$ aws configure
```

When the response gets to `Default output format`, enter **`json`**.

Uninstall the CloudWatch agent with Prometheus monitoring

When you are finished testing, enter the following command to uninstall the CloudWatch agent by deleting the Amazon CloudFormation stack.

```
aws cloudformation delete-stack \
--stack-name CWAgent-Prometheus-ECS-${ECS_CLUSTER_NAME}-EC2-${ECS_NETWORK_MODE} \
--region ${AWS_DEFAULT_REGION} \
--profile ${AWS_PROFILE}
```

Sample Java/JMX workload for Amazon ECS clusters

JMX Exporter is an official Prometheus exporter that can scrape and expose JMX mBeans as Prometheus metrics. For more information, see prometheus/jmx_exporter.

The CloudWatch agent with Prometheus support scrapes the Java/JMX Prometheus metrics based on the service discovery configuration in the Amazon ECS cluster. You can configure the JMX Exporter to expose the metrics on a different port or metrics_path. If you do change the port or path, update the default `ecs_service_discovery` section in the CloudWatch agent configuration.

To collect metrics from a sample Prometheus workload for Amazon ECS, you must be running Container Insights in the cluster. For information about installing Container Insights, see Setting up Container Insights on Amazon ECS.

To install the Java/JMX sample workload for Amazon ECS clusters

1. Follow the steps in these sections to create your Docker images.

   - Example: Java Jar Application Docker image with Prometheus metrics
   - Example: Apache Tomcat Docker image with Prometheus metrics
2. Specify the following two docker labels in the Amazon ECS task definition file. You can then run the task definition as an Amazon ECS service or Amazon ECS task in the cluster.

   - Set `ECS_PROMETHEUS_EXPORTER_PORT` to point to the containerPort where the Prometheus metrics are exposed.
   - Set `Java_EMF_Metrics` to `true`. The CloudWatch agent uses this flag to generated the embedded metric format in the log event.

   The following is an example:

```
{
  "family": "workload-java-ec2-bridge",
  "taskRoleArn": "{{task-role-arn}}",
  "executionRoleArn": "{{execution-role-arn}}",
  "networkMode": "bridge",
  "containerDefinitions": [
    {
      "name": "tomcat-prometheus-workload-java-ec2-bridge-dynamic-port",
      "image": "your_docker_image_tag_for_tomcat_with_prometheus_metrics",
      "portMappings": [
        {
          "hostPort": 0,
          "protocol": "tcp",
          "containerPort": 9404
```

```
        }
      ],
      "dockerLabels": {
        "ECS_PROMETHEUS_EXPORTER_PORT": "9404",
        "Java_EMF_Metrics": "true"
      }
    }
  ],
  "requiresCompatibilities": [
    "EC2"  ],
  "cpu": "256",
  "memory": "512"
  }
```

The default setting of the CloudWatch agent in the Amazon CloudFormation template enables both docker label-based service discovery and task definition ARN-based service discovery. To view these default settings, see line 65 of the CloudWatch agent YAML configuration file. The containers with the `ECS_PROMETHEUS_EXPORTER_PORT` label will be auto-discovered based on the specified container port for Prometheus scraping.

The default setting of the CloudWatch agent also has the `metric_declaration` setting for Java/JMX at line 112 of the same file. All docker labels of the target containers will be added as additional labels in the Prometheus metrics and sent to CloudWatch Logs. For the Java/JMX containers with docker label `Java_EMF_Metrics="true"`, the embedded metric format will be generated.

Sample NGINX workload for Amazon ECS clusters

The NGINX Prometheus exporter can scrape and expose NGINX data as Prometheus metrics. This example uses the exporter in tandem with the NGINX reverse proxy service for Amazon ECS.

For more information about the NGINX Prometheus exporter, see nginx-prometheus-exporter on Github. For more information about the NGINX reverse proxy, see ecs-nginx-reverse-proxy on Github.

The CloudWatch agent with Prometheus support scrapes the NGINX Prometheus metrics based on the service discovery configuration in the Amazon ECS cluster. You can configure the NGINX Prometheus Exporter to expose the metrics on a different port or path. If you change the port or path, update the `ecs_service_discovery` section in the CloudWatch agent configuration file.

Install the NGINX reverse proxy sample workload for Amazon ECS clusters

Follow these steps to install the NGINX reverse proxy sample workload.

Create the Docker images

To create the Docker images for the NGINX reverse proxy sample workload

1. Download the following folder from the NGINX reverse proxy repo: https://github.com/awslabs/ecs-nginx-reverse-proxy/tree/master/reverse-proxy/.
2. Find the `app` directory and build an image from that directory:

   ```
   docker build -t web-server-app ./path-to-app-directory
   ```

3. Build a custom image for NGINX. First, create a directory with the following two files:

   - A sample Dockerfile:

     ```
     FROM nginx
     COPY nginx.conf /etc/nginx/nginx.conf
     ```

   - An `nginx.conf` file, modified from https://github.com/awslabs/ecs-nginx-reverse-proxy/tree/master/reverse-proxy/:

     ```
     events {
       worker_connections 768;
     }

     http {
       # Nginx will handle gzip compression of responses from the app server
       gzip on;
       gzip_proxied any;
       gzip_types text/plain application/json;
       gzip_min_length 1000;

       server{
         listen 8080;
         location /stub_status {
             stub_status   on;
         }
       }
     ```

```
    server {
      listen 80;

      # Nginx will reject anything not matching /api
      location /api {
        # Reject requests with unsupported HTTP method
        if ($request_method !~ ^(GET|POST|HEAD|OPTIONS|PUT|DELETE)$) {
          return 405;
        }

        # Only requests matching the whitelist expectations will
        # get sent to the application server
        proxy_pass http://app:3000;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_cache_bypass $http_upgrade;
      }
    }
  }
```

Note

`stub_status` must be enabled on the same port that `nginx-prometheus-exporter` is configured to scrape metrics from. In our example task definition, `nginx-prometheus-exporter` is configured to scrape metrics from port 8080.

4. Build an image from files in your new directory:

```
docker build -t nginx-reverse-proxy ./path-to-your-directory
```

5. Upload your new images to an image repository for later use.

Create the task definition to run NGINX and the web server app in Amazon ECS

Next, you set up the task definition.

This task definition enables the collection and export of NGINX Prometheus metrics. The NGINX container tracks input from the app, and exposes that data to port 8080, as set in `nginx.conf`.

The NGINX prometheus exporter container scrapes these metrics, and posts them to port 9113, for use in CloudWatch.

To set up the task definition for the NGINX sample Amazon ECS workload

1. Create a task definition JSON file with the following content. Replace *your-customized-nginx-iamge* with the image URI for your customized NGINX image, and replace *your-web-server-app-image* with the image URI for your web server app image.

```
{
  "containerDefinitions": [
    {
      "name": "nginx",
      "image": "your-customized-nginx-image",
      "memory": 256,
      "cpu": 256,
      "essential": true,
      "portMappings": [
        {
          "containerPort": 80,
          "protocol": "tcp"
        }
      ],
      "links": [
        "app"
      ]
    },
    {
      "name": "app",
      "image": "your-web-server-app-image",
      "memory": 256,
      "cpu": 256,
      "essential": true
    },
    {
      "name": "nginx-prometheus-exporter",
      "image": "docker.io/nginx/nginx-prometheus-exporter:0.8.0",
      "memory": 256,
      "cpu": 256,
      "essential": true,
      "command": [
        "-nginx.scrape-uri",
        "http://nginx:8080/stub_status"
```

```
    ],
    "links":[
      "nginx"
    ],
      "portMappings":[
        {
          "containerPort": 9113,
          "protocol": "tcp"
        }
      ]
    }
  ],
  "networkMode": "bridge",
  "placementConstraints": [],
  "family": "nginx-sample-stack"
}
```

2. Register the task definition by entering the following command.

```
aws ecs register-task-definition --cli-input-json file://path-to-your-task-definition-json
```

3. Create a service to run the task by entering the following command:

   Be sure not to change the service name. We will be running a CloudWatch agent service using a configuration that searches for tasks using the name patterns of the services that started them. For example, for the CloudWatch agent to find the task launched by this command, you can specify the value of `sd_service_name_pattern` to be `^nginx-service$`. The next section provides more details.

```
aws ecs create-service \
 --cluster your-cluster-name \
 --service-name nginx-service \
 --task-definition nginx-sample-stack:1 \
 --desired-count 1
```

Configure the CloudWatch agent to scrape NGINX Prometheus metrics

The final step is to configure the CloudWatch agent to scrape the NGINX metrics. In this example, the CloudWatch agent discovers the task via the service name pattern, and the port 9113, where the exporter exposes the prometheus metrics for NGINX. With the task discovered and the metrics available, the CloudWatch agent begins posting the collected metrics to the log stream **nginx-prometheus-exporter**.

To configure the CloudWatch agent to scrape the NGINX metrics

1. Download the latest version of the necessary YAML file by entering the following command.

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/ecs-task-definition-templates/deployment-mode/replica-service/cwagent-prometheus/cloudformation-quickstart/cwagent-ecs-prometheus-metric-for-bridge-host.yaml
```

2. Open the file with a text editor, and find the full CloudWatch agent confguration in the `value` key in the `resource:CWAgentConfigSSMParameter` section. Then, in the `ecs_service_discovery` section, add the following `service_name_list_for_tasks` section.

```
"service_name_list_for_tasks": [
  {
    "sd_job_name": "nginx-prometheus-exporter",
    "sd_metrics_path": "/metrics",
    "sd_metrics_ports": "9113",
    "sd_service_name_pattern": "^nginx-service$"
   }
],
```

3. In the same file, add the following section in the `metric_declaration` section to allow NGINX metrics. Be sure to follow the existing indentation pattern.

```
{
  "source_labels": ["job"],
  "label_matcher": ".*nginx.*",
  "dimensions": [["ClusterName", "TaskDefinitionFamily", "ServiceName"]],
  "metric_selectors": [
    "^nginx_.*$"
  ]
},
```

4. If you don't already have the CloudWatch agent deployed in this cluster, skip to step 8.

   If you already have the CloudWatch agent deployed in the Amazon ECS cluster by using Amazon CloudFormation, you can create a change set by entering the following commands:

```
ECS_CLUSTER_NAME=your_cluster_name
AWS_REGION=your_aws_region
ECS_NETWORK_MODE=bridge
CREATE_IAM_ROLES=True
ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name

aws cloudformation create-change-set --stack-name CWAgent-Prometheus-ECS-
${ECS_CLUSTER_NAME}-EC2-${ECS_NETWORK_MODE} \
    --template-body file://cwagent-ecs-prometheus-metric-for-bridge-host.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=$ECS_CLUSTER_NAME \
                 ParameterKey=CreateIAMRoles,ParameterValue=$CREATE_IAM_ROLES \
                 ParameterKey=ECSNetworkMode,ParameterValue=$ECS_NETWORK_MODE \
                 ParameterKey=TaskRoleName,ParameterValue=$ECS_TASK_ROLE_NAME \
                 ParameterKey=ExecutionRoleName,ParameterValue=
$ECS_EXECUTION_ROLE_NAME \
    --capabilities CAPABILITY_NAMED_IAM \
    --region $AWS_REGION \
    --change-set-name nginx-scraping-support
```

5. Open the Amazon CloudFormation console at https://console.amazonaws.cn/cloudformation.

6. Revew the newly-created changeset **nginx-scraping-support**. You should see one change applied to the **CWAgentConfigSSMParameter** resource. Run the changeset and restart the CloudWatch agent task by entering the following command:

```
aws ecs update-service --cluster $ECS_CLUSTER_NAME \
--desired-count 0 \
--service cwagent-prometheus-replica-service-EC2-$ECS_NETWORK_MODE \
--region $AWS_REGION
```

7. Wait about 10 seconds, and then enter the following command.

```
aws ecs update-service --cluster $ECS_CLUSTER_NAME \
--desired-count 1 \
--service cwagent-prometheus-replica-service-EC2-$ECS_NETWORK_MODE \
--region $AWS_REGION
```

8. If you are installing the CloudWatch agent with Prometheus metric collecting on the cluster for the first time, enter the following commands.

```
ECS_CLUSTER_NAME=your_cluster_name
```

```
AWS_REGION=your_aws_region
ECS_NETWORK_MODE=bridge
CREATE_IAM_ROLES=True
ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name

aws cloudformation create-stack --stack-name CWAgent-Prometheus-ECS-
${ECS_CLUSTER_NAME}-EC2-${ECS_NETWORK_MODE} \
    --template-body file://cwagent-ecs-prometheus-metric-for-bridge-host.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=$ECS_CLUSTER_NAME \
                 ParameterKey=CreateIAMRoles,ParameterValue=$CREATE_IAM_ROLES \
                 ParameterKey=ECSNetworkMode,ParameterValue=$ECS_NETWORK_MODE \
                 ParameterKey=TaskRoleName,ParameterValue=$ECS_TASK_ROLE_NAME \
                 ParameterKey=ExecutionRoleName,ParameterValue=
$ECS_EXECUTION_ROLE_NAME \
    --capabilities CAPABILITY_NAMED_IAM \
    --region $AWS_REGION
```

Viewing your NGINX metrics and logs

You can now view the NGINX metrics being collected.

To view the metrics for your sample NGINX workload

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the Region where your cluster is running, choose **Metrics** in the left navigation pane. Find the **ContainerInsights/Prometheus** namespace to see the metrics.
3. To see the CloudWatch Logs events, choose **Log groups** in the navigation pane. The events are in the log group **/aws/containerinsights/*your_cluster_name*/prometheus**, in the log stream *nginx-prometheus-exporter*.

Sample NGINX Plus workload for Amazon ECS clusters

NGINX Plus is the commerical version of NGINX. You must have a licence to use it. For more information, see NGINX Plus

The NGINX Prometheus exporter can scrape and expose NGINX data as Prometheus metrics. This example uses the exporter in tandem with the NGINX Plus reverse proxy service for Amazon ECS.

For more information about the NGINX Prometheus exporter, see [nginx-prometheus-exporter] on Github. For more information about the NGINX reverse proxy, see [ecs-nginx-reverse-proxy] on Github.

The CloudWatch agent with Prometheus support scrapes the NGINX Plus Prometheus metrics based on the service discovery configuration in the Amazon ECS cluster. You can configure the NGINX Prometheus Exporter to expose the metrics on a different port or path. If you change the port or path, update the `ecs_service_discovery` section in the CloudWatch agent configuration file.

Install the NGINX Plus reverse proxy sample workload for Amazon ECS clusters

Follow these steps to install the NGINX reverse proxy sample workload.

Create the Docker images

To create the Docker images for the NGINX Plus reverse proxy sample workload

1. Download the following folder from the NGINX reverse proxy repo: https://github.com/awslabs/ecs-nginx-reverse-proxy/tree/master/reverse-proxy/.
2. Find the app directory and build an image from that directory:

```
docker build -t web-server-app ./path-to-app-directory
```

3. Build a custom image for NGINX Plus. Before you can build the image for NGINX Plus, you need to obtain the key named `nginx-repo.key` and the SSL certificate `nginx-repo.crt` for your licensed NGINX Plus. Create a directory and store in it your `nginx-repo.key` and `nginx-repo.crt` files.

   In the directory that you just created, create the following two files:

   - A sample Dockerfile with the following content. This docker file is adopted from a sample file provided at https://docs.nginx.com/nginx/admin-guide/installing-nginx/installing-nginx-docker/#docker_plus_image. The important change that we make is that we load a separate file, called `nginx.conf`, which will be created in the next step.

```
FROM debian:buster-slim

LABEL maintainer="NGINX Docker Maintainers <docker-maint@nginx.com>“

# Define NGINX versions for NGINX Plus and NGINX Plus modules
```

```
# Uncomment this block and the versioned nginxPackages block in the main RUN
# instruction to install a specific release
# ENV NGINX_VERSION 21
# ENV NJS_VERSION 0.3.9
# ENV PKG_RELEASE 1~buster

# Download certificate and key from the customer portal (https://cs.nginx.com
 (https://cs.nginx.com/))
# and copy to the build context
COPY nginx-repo.crt /etc/ssl/nginx/
COPY nginx-repo.key /etc/ssl/nginx/
# COPY nginx.conf /etc/ssl/nginx/nginx.conf

RUN set -x \
# Create nginx user/group first, to be consistent throughout Docker variants
&& addgroup --system --gid 101 nginx \
&& adduser --system --disabled-login --ingroup nginx --no-create-home --home /
nonexistent --gecos "nginx user" --shell /bin/false --uid 101 nginx \
&& apt-get update \
&& apt-get install --no-install-recommends --no-install-suggests -y ca-
certificates gnupg1 \
&& \
NGINX_GPGKEY=573BFD6B3D8FBC641079A6ABABF5BD827BD9BF62; \
found=''; \
for server in \
ha.pool.sks-keyservers.net (http://ha.pool.sks-keyservers.net/) \
hkp://keyserver.ubuntu.com:80 \
hkp://p80.pool.sks-keyservers.net:80 \
pgp.mit.edu (http://pgp.mit.edu/) \
; do \
echo "Fetching GPG key $NGINX_GPGKEY from $server"; \
apt-key adv --keyserver "$server" --keyserver-options timeout=10 --recv-keys
 "$NGINX_GPGKEY" && found=yes && break; \
done; \
test -z "$found" && echo >&2 "error: failed to fetch GPG key $NGINX_GPGKEY" &&
 exit 1; \
apt-get remove --purge --auto-remove -y gnupg1 && rm -rf /var/lib/apt/lists/* \
# Install the latest release of NGINX Plus and/or NGINX Plus modules
# Uncomment individual modules if necessary
# Use versioned packages over defaults to specify a release
&& nginxPackages=" \
nginx-plus \
# nginx-plus=${NGINX_VERSION}-${PKG_RELEASE} \
# nginx-plus-module-xslt \
```

```
# nginx-plus-module-xslt=${NGINX_VERSION}-${PKG_RELEASE} \
# nginx-plus-module-geoip \
# nginx-plus-module-geoip=${NGINX_VERSION}-${PKG_RELEASE} \
# nginx-plus-module-image-filter \
# nginx-plus-module-image-filter=${NGINX_VERSION}-${PKG_RELEASE} \
# nginx-plus-module-perl \
# nginx-plus-module-perl=${NGINX_VERSION}-${PKG_RELEASE} \
# nginx-plus-module-njs \
# nginx-plus-module-njs=${NGINX_VERSION}+${NJS_VERSION}-${PKG_RELEASE} \
" \
&& echo "Acquire::https::plus-pkgs.nginx.com::Verify-Peer \"true\";" >> /etc/apt/
apt.conf.d/90nginx \
&& echo "Acquire::https::plus-pkgs.nginx.com::Verify-Host \"true\";" >> /etc/apt/
apt.conf.d/90nginx \
&& echo "Acquire::https::plus-pkgs.nginx.com::SslCert \"/etc/ssl/nginx/nginx-
repo.crt\";" >> /etc/apt/apt.conf.d/90nginx \
&& echo "Acquire::https::plus-pkgs.nginx.com::SslKey \"/etc/ssl/nginx/nginx-
repo.key\";" >> /etc/apt/apt.conf.d/90nginx \
&& printf "deb https://plus-pkgs.nginx.com/debian buster nginx-plus\n" > /etc/
apt/sources.list.d/nginx-plus.list \
&& apt-get update \
&& apt-get install --no-install-recommends --no-install-suggests -y \
$nginxPackages \
gettext-base \
curl \
&& apt-get remove --purge --auto-remove -y && rm -rf /var/lib/apt/lists/* /etc/
apt/sources.list.d/nginx-plus.list \
&& rm -rf /etc/apt/apt.conf.d/90nginx /etc/ssl/nginx

# Forward request logs to Docker log collector
RUN ln -sf /dev/stdout /var/log/nginx/access.log \
&& ln -sf /dev/stderr /var/log/nginx/error.log

COPY nginx.conf /etc/nginx/nginx.conf

EXPOSE 80

STOPSIGNAL SIGTERM

CMD ["nginx", "-g", "daemon off;"]
```

- An `nginx.conf` file, modified from https://github.com/awslabs/ecs-nginx-reverse-proxy/tree/master/reverse-proxy/nginx.

```
events {
  worker_connections 768;
}

http {
  # Nginx will handle gzip compression of responses from the app server
  gzip on;
  gzip_proxied any;
  gzip_types text/plain application/json;
  gzip_min_length 1000;

  upstream backend {
    zone name 10m;
    server app:3000    weight=2;
    server app2:3000    weight=1;
  }

  server{
    listen 8080;
    location /api {
      api write=on;
    }
  }

  match server_ok {
    status 100-599;
  }

  server {
    listen 80;
    status_zone zone;
    # Nginx will reject anything not matching /api
    location /api {
      # Reject requests with unsupported HTTP method
      if ($request_method !~ ^(GET|POST|HEAD|OPTIONS|PUT|DELETE)$) {
        return 405;
      }

      # Only requests matching the whitelist expectations will
      # get sent to the application server
      proxy_pass http://backend;
      health_check uri=/lorem-ipsum match=server_ok;
      proxy_http_version 1.1;
```

```
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_cache_bypass $http_upgrade;
      }
    }
  }
```

4. Build an image from files in your new directory:

```
docker build -t nginx-plus-reverse-proxy ./path-to-your-directory
```

5. Upload your new images to an image repository for later use.

Create the task definition to run NGINX Plus and the web server app in Amazon ECS

Next, you set up the task definition.

This task definition enables the collection and export of NGINX Plus Prometheus metrics. The NGINX container tracks input from the app, and exposes that data to port 8080, as set in `nginx.conf`. The NGINX prometheus exporter container scrapes these metrics, and posts them to port 9113, for use in CloudWatch.

To set up the task definition for the NGINX sample Amazon ECS workload

1. Create a task definition JSON file with the following content. Replace *your-customized-nginx-plus-image* with the image URI for your customized NGINX Plus image, and replace *your-web-server-app-image* with the image URI for your web server app image.

```
{
  "containerDefinitions": [
    {
      "name": "nginx",
      "image": "your-customized-nginx-plus-image",
      "memory": 256,
      "cpu": 256,
      "essential": true,
      "portMappings": [
        {
          "containerPort": 80,
          "protocol": "tcp"
```

```
        }
      ],
      "links": [
        "app",
        "app2"
      ]
    },
    {
      "name": "app",
      "image": "your-web-server-app-image",
      "memory": 256,
      "cpu": 128,
      "essential": true
    },
    {
      "name": "app2",
      "image": "your-web-server-app-image",
      "memory": 256,
      "cpu": 128,
      "essential": true
    },
    {
      "name": "nginx-prometheus-exporter",
      "image": "docker.io/nginx/nginx-prometheus-exporter:0.8.0",
      "memory": 256,
      "cpu": 256,
      "essential": true,
      "command": [
        "-nginx.plus",
        "-nginx.scrape-uri",
         "http://nginx:8080/api"
    ],
    "links":[
      "nginx"
    ],
      "portMappings":[
        {
          "containerPort": 9113,
          "protocol": "tcp"
        }
      ]
    }
  ],
  "networkMode": "bridge",
```

```
  "placementConstraints": [],
  "family": "nginx-plus-sample-stack"
}
```

2. Register the task definition:

```
aws ecs register-task-definition --cli-input-json file://path-to-your-task-definition-json
```

3. Create a service to run the task by entering the following command:

```
aws ecs create-service \
 --cluster your-cluster-name \
 --service-name nginx-plus-service \
 --task-definition nginx-plus-sample-stack:1 \
 --desired-count 1
```

   Be sure not to change the service name. We will be running a CloudWatch agent service using a configuration that searches for tasks using the name patterns of the services that started them. For example, for the CloudWatch agent to find the task launched by this command, you can specify the value of `sd_service_name_pattern` to be `^nginx-plus-service$`. The next section provides more details.

Configure the CloudWatch agent to scrape NGINX Plus Prometheus metrics

The final step is to configure the CloudWatch agent to scrape the NGINX metrics. In this example, the CloudWatch agent discovers the task via the service name pattern, and the port 9113, where the exporter exposes the prometheus metrics for NGINX. With the task discovered and the metrics available, the CloudWatch agent begins posting the collected metrics to the log stream **nginx-prometheus-exporter**.

To configure the CloudWatch agent to scrape the NGINX metrics

1. Download the latest version of the necessary YAML file by entering the following command.

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/ecs-task-definition-templates/deployment-mode/replica-service/cwagent-prometheus/cloudformation-quickstart/cwagent-ecs-prometheus-metric-for-bridge-host.yaml
```

2. Open the file with a text editor, and find the full CloudWatch agent confguration in the `value` key in the `resource:CWAgentConfigSSMParameter` section. Then, in the `ecs_service_discovery` section, add the following `service_name_list_for_tasks` section.

```
"service_name_list_for_tasks": [
  {
    "sd_job_name": "nginx-plus-prometheus-exporter",
    "sd_metrics_path": "/metrics",
    "sd_metrics_ports": "9113",
    "sd_service_name_pattern": "^nginx-plus.*"
   }
],
```

3. In the same file, add the following section in the `metric_declaration` section to allow NGINX Plus metrics. Be sure to follow the existing indentation pattern.

```
{
  "source_labels": ["job"],
  "label_matcher": "^nginx-plus.*",
  "dimensions": [["ClusterName", "TaskDefinitionFamily", "ServiceName"]],
  "metric_selectors": [
    "^nginxplus_connections_accepted$",
    "^nginxplus_connections_active$",
    "^nginxplus_connections_dropped$",
    "^nginxplus_connections_idle$",
    "^nginxplus_http_requests_total$",
    "^nginxplus_ssl_handshakes$",
    "^nginxplus_ssl_handshakes_failed$",
    "^nginxplus_up$",
    "^nginxplus_upstream_server_health_checks_fails$"
  ]
},
{
  "source_labels": ["job"],
  "label_matcher": "^nginx-plus.*",
  "dimensions": [["ClusterName", "TaskDefinitionFamily", "ServiceName",
 "upstream"]],
  "metric_selectors": [
    "^nginxplus_upstream_server_response_time$"
  ]
},
```

```
{
  "source_labels": ["job"],
  "label_matcher": "^nginx-plus.*",
  "dimensions": [["ClusterName", "TaskDefinitionFamily", "ServiceName", "code"]],
  "metric_selectors": [
    "^nginxplus_upstream_server_responses$",
    "^nginxplus_server_zone_responses$"
  ]
},
```

4. If you don't already have the CloudWatch agent deployed in this cluster, skip to step 8.

   If you already have the CloudWatch agent deployed in the Amazon ECS cluster by using Amazon CloudFormation, you can create a change set by entering the following commands:

```
ECS_CLUSTER_NAME=your_cluster_name
AWS_REGION=your_aws_region
ECS_NETWORK_MODE=bridge
CREATE_IAM_ROLES=True
ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name

aws cloudformation create-change-set --stack-name CWAgent-Prometheus-ECS-
${ECS_CLUSTER_NAME}-EC2-${ECS_NETWORK_MODE} \
    --template-body file://cwagent-ecs-prometheus-metric-for-bridge-host.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=$ECS_CLUSTER_NAME \
                 ParameterKey=CreateIAMRoles,ParameterValue=$CREATE_IAM_ROLES \
                 ParameterKey=ECSNetworkMode,ParameterValue=$ECS_NETWORK_MODE \
                 ParameterKey=TaskRoleName,ParameterValue=$ECS_TASK_ROLE_NAME \
                 ParameterKey=ExecutionRoleName,ParameterValue=
$ECS_EXECUTION_ROLE_NAME \
    --capabilities CAPABILITY_NAMED_IAM \
    --region $AWS_REGION \
    --change-set-name nginx-plus-scraping-support
```

5. Open the Amazon CloudFormation console at https://console.amazonaws.cn/cloudformation.

6. Revew the newly-created changeset **nginx-plus-scraping-support**. You should see one change applied to the **CWAgentConfigSSMParameter** resource. Run the changeset and restrt the CloudWatch agent task by entering the following command:

```
aws ecs update-service --cluster $ECS_CLUSTER_NAME \
--desired-count 0 \
```

```
--service cwagent-prometheus-replica-service-EC2-$ECS_NETWORK_MODE \
--region $AWS_REGION
```

7. Wait about 10 seconds, and then enter the following command.

```
aws ecs update-service --cluster $ECS_CLUSTER_NAME \
--desired-count 1 \
--service cwagent-prometheus-replica-service-EC2-$ECS_NETWORK_MODE \
--region $AWS_REGION
```

8. If you are installing the CloudWatch agent with Prometheus metric collecting on the cluster for the first time, enter the following commands.

```
ECS_CLUSTER_NAME=your_cluster_name
AWS_REGION=your_aws_region
ECS_NETWORK_MODE=bridge
CREATE_IAM_ROLES=True
ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name

aws cloudformation create-stack --stack-name CWAgent-Prometheus-ECS-
${ECS_CLUSTER_NAME}-EC2-${ECS_NETWORK_MODE} \
    --template-body file://cwagent-ecs-prometheus-metric-for-bridge-host.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=$ECS_CLUSTER_NAME \
                 ParameterKey=CreateIAMRoles,ParameterValue=$CREATE_IAM_ROLES \
                 ParameterKey=ECSNetworkMode,ParameterValue=$ECS_NETWORK_MODE \
                 ParameterKey=TaskRoleName,ParameterValue=$ECS_TASK_ROLE_NAME \
                 ParameterKey=ExecutionRoleName,ParameterValue=
$ECS_EXECUTION_ROLE_NAME \
    --capabilities CAPABILITY_NAMED_IAM \
    --region $AWS_REGION
```

Viewing your NGINX Plus metrics and logs

You can now view the NGINX Plus metrics being collected.

To view the metrics for your sample NGINX workload

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the Region where your cluster is running, choose **Metrics** in the left navigation pane. Find the **ContainerInsights/Prometheus** namespace to see the metrics.

3. To see the CloudWatch Logs events, choose **Log groups** in the navigation pane. The events are in the log group **/aws/containerinsights/*your_cluster_name*/prometheus**, in the log stream *nginx-plus-prometheus-exporter*.

Tutorial for adding a new Prometheus scrape target: Memcached on Amazon ECS

This tutorial provides a hands-on introduction to scrape the Prometheus metrics of a sample Memcached application on an Amazon Amazon ECS cluster with the EC2 launch type. The Memcached Prometheus exporter target will be auto-discovered by the CloudWatch agent by ECS task definition-based service discovery.

Memcached is a general-purpose distributed memory caching system. It is often used to speed up dynamic database-driven websites by caching data and objects in RAM to reduce the number of times an external data source (such as a database or API) must be read. For more infromation, see [What is Memcached?](#)

The [memchached_exporter](#) (Apache License 2.0) is one of the Prometheus official exporters. By default the memcache_exporter serves on port 0.0.0.0:9150 at `/metrics.`

The Docker images in the following two Docker Hub repositories are used in this tutorial:

- [Memcached](#)
- [prom/memcached-exporter](#)

Prerequisite

To collect metrics from a sample Prometheus workload for Amazon ECS, you must be running Container Insights in the cluster. For information about installing Container Insights, see [Setting up Container Insights on Amazon ECS](#).

Topics

- [Set the Amazon ECS EC2 cluster environment variables](#)
- [Install the sample Memcached workload](#)
- [Configure the CloudWatch agent to scrape Memcached Prometheus metrics](#)
- [Viewing your Memcached metrics](#)

Set the Amazon ECS EC2 cluster environment variables

To set the Amazon ECS EC2 cluster environment variables

1. Install the Amazon ECS CLI if you haven't already done so. For more information, see [Installing the Amazon ECS CLI](#).
2. Set the new Amazon ECS cluster name and Region. For example:

```
ECS_CLUSTER_NAME=ecs-ec2-memcached-tutorial
AWS_DEFAULT_REGION=ca-central-1
```

3. (Optional) If you don't already have an Amazon ECS cluster with the EC2 launch type where you want to install the sample Memcached workload and CloudWatch agent, you can create one by entering the following command.

```
ecs-cli up --capability-iam --size 1 \
--instance-type t3.medium \
--cluster $ECS_CLUSTER_NAME \
--region $AWS_REGION
```

The expected result of this command is as follows:

```
WARN[0000] You will not be able to SSH into your EC2 instances without a key pair.
INFO[0000] Using recommended Amazon Linux 2 AMI with ECS Agent 1.44.4 and Docker
 version 19.03.6-ce
INFO[0001] Created cluster                               cluster=ecs-ec2-memcached-
tutorial region=ca-central-1
INFO[0002] Waiting for your cluster resources to be created...
INFO[0002] Cloudformation stack status
 stackStatus=CREATE_IN_PROGRESS
INFO[0063] Cloudformation stack status
 stackStatus=CREATE_IN_PROGRESS
INFO[0124] Cloudformation stack status
 stackStatus=CREATE_IN_PROGRESS
VPC created: vpc-xxxxxxxxxxxxxxxxx
Security Group created: sg-xxxxxxxxxxxxxxxxx
Subnet created: subnet-xxxxxxxxxxxxxxxx
Subnet created: subnet-xxxxxxxxxxxxxxxx
Cluster creation succeeded.
```

Install the sample Memcached workload

To install the sample Memcached workload which exposes the Prometheus metrics

1. Download the Memcached Amazon CloudFormation template by entering the following command.

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/ecs-task-definition-templates/deployment-mode/replica-service/cwagent-prometheus/sample_traffic/memcached/memcached-traffic-sample.yaml
```

2. Set the IAM role names to be created for Memcached by entering the following commands.

```
MEMCACHED_ECS_TASK_ROLE_NAME=memcached-prometheus-demo-ecs-task-role-name
MEMCACHED_ECS_EXECUTION_ROLE_NAME=memcached-prometheus-demo-ecs-execution-role-name
```

3. Install the sample Memcached workload by entering the following command. This sample installs the workload in `host` network mode.

```
MEMCACHED_ECS_NETWORK_MODE=host

aws cloudformation create-stack --stack-name Memcached-Prometheus-Demo-ECS-$ECS_CLUSTER_NAME-EC2-$MEMCACHED_ECS_NETWORK_MODE \
    --template-body file://memcached-traffic-sample.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=$ECS_CLUSTER_NAME \
                 ParameterKey=ECSNetworkMode,ParameterValue=$MEMCACHED_ECS_NETWORK_MODE \
                 ParameterKey=TaskRoleName,ParameterValue=$MEMCACHED_ECS_TASK_ROLE_NAME \
                 ParameterKey=ExecutionRoleName,ParameterValue=$MEMCACHED_ECS_EXECUTION_ROLE_NAME \
    --capabilities CAPABILITY_NAMED_IAM \
    --region $AWS_REGION
```

The Amazon CloudFormation stack creates four resources:

- One ECS task role
- One ECS task execution role
- One Memcached task definition
- One Memcached service

In the Memcached task definition, two containers are defined:

- The primary container runs a simple Memcached application and opens port 11211 for access.
- The other container runs the Redis OSS exporter process to expose the Prometheus metrics on port 9150. This is the container to be discovered and scraped by the CloudWatch agent.

Configure the CloudWatch agent to scrape Memcached Prometheus metrics

To configure the CloudWatch agent to scrape Memcached Prometheus metrics

1. Download the latest version of `cwagent-ecs-prometheus-metric-for-awsvpc.yaml` by entering the following command.

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
insights/latest/ecs-task-definition-templates/deployment-mode/replica-service/
cwagent-prometheus/cloudformation-quickstart/cwagent-ecs-prometheus-metric-for-
awsvpc.yaml
```

2. Open the file with a text editor, and find the full CloudWatch agent configuration behind the `value` key in the `resource:CWAgentConfigSSMParameter` section.

   Then, in the `ecs_service_discovery` section, add the following configuration into the `task_definition_list` section.

```
{
    "sd_job_name": "ecs-memcached",
    "sd_metrics_ports": "9150",
    "sd_task_definition_arn_pattern": ".*:task-definition/memcached-prometheus-
demo.*:[0-9]+"
},
```

   For the `metric_declaration` section, the default setting does not allow any Memcached metrics. Add the following section to allow Memcached metrics. Be sure to follow the existing indentation pattern.

```
{
  "source_labels": ["container_name"],
  "label_matcher": "memcached-exporter-.*",
  "dimensions": [["ClusterName", "TaskDefinitionFamily"]],
  "metric_selectors": [
```

```
      "^memcached_current_(bytes|items|connections)$",
      "^memcached_items_(reclaimed|evicted)_total$",
      "^memcached_(written|read)_bytes_total$",
      "^memcached_limit_bytes$",
      "^memcached_commands_total$"
    ]
  },
  {
    "source_labels": ["container_name"],
    "label_matcher": "memcached-exporter-.*",
    "dimensions": [["ClusterName", "TaskDefinitionFamily","status","command"],
   ["ClusterName", "TaskDefinitionFamily","command"]],
    "metric_selectors": [
      "^memcached_commands_total$"
    ]
  },
```

3. If you already have the CloudWatch agent deployed in the Amazon ECS cluster by Amazon CloudFormation, you can create a change set by entering the following commands.

```
ECS_NETWORK_MODE=bridge
CREATE_IAM_ROLES=True
ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name

aws cloudformation create-change-set --stack-name CWAgent-Prometheus-ECS-
${ECS_CLUSTER_NAME}-EC2-${ECS_NETWORK_MODE} \
    --template-body file://cwagent-ecs-prometheus-metric-for-bridge-host.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=$ECS_CLUSTER_NAME \
                 ParameterKey=CreateIAMRoles,ParameterValue=$CREATE_IAM_ROLES \
                 ParameterKey=ECSNetworkMode,ParameterValue=$ECS_NETWORK_MODE \
                 ParameterKey=TaskRoleName,ParameterValue=$ECS_TASK_ROLE_NAME \
                 ParameterKey=ExecutionRoleName,ParameterValue=
$ECS_EXECUTION_ROLE_NAME \
    --capabilities CAPABILITY_NAMED_IAM \
    --region $AWS_REGION \
    --change-set-name memcached-scraping-support
```

4. Open the Amazon CloudFormation console at https://console.amazonaws.cn/cloudformation.
5. Review the newly created changeset `memcached-scraping-support`. You should see one change applied to the `CWAgentConfigSSMParameter` resource. Execute the changeset and restart the CloudWatch agent task by entering the following commands.

```
aws ecs update-service --cluster $ECS_CLUSTER_NAME \
--desired-count 0 \
--service cwagent-prometheus-replica-service-EC2-$ECS_NETWORK_MODE \
--region $AWS_REGION
```

6. Wait about 10 seconds, and then enter the following command.

```
aws ecs update-service --cluster $ECS_CLUSTER_NAME \
--desired-count 1 \
--service cwagent-prometheus-replica-service-EC2-$ECS_NETWORK_MODE \
--region $AWS_REGION
```

7. If you are installing the CloudWatch agent with Prometheus metric collecting for the cluster for the first time, please enter the following commands:

```
ECS_NETWORK_MODEE=bridge
CREATE_IAM_ROLES=True
ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name

aws cloudformation create-stack --stack-name CWAgent-Prometheus-ECS-
${ECS_CLUSTER_NAME}-EC2-${ECS_NETWORK_MODE} \
    --template-body file://cwagent-ecs-prometheus-metric-for-bridge-host.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=$ECS_CLUSTER_NAME \
                 ParameterKey=CreateIAMRoles,ParameterValue=$CREATE_IAM_ROLES \
                 ParameterKey=ECSNetworkMode,ParameterValue=$ECS_NETWORK_MODE \
                 ParameterKey=TaskRoleName,ParameterValue=$ECS_TASK_ROLE_NAME \
                 ParameterKey=ExecutionRoleName,ParameterValue=
$ECS_EXECUTION_ROLE_NAME \
    --capabilities CAPABILITY_NAMED_IAM \
    --region $AWS_REGION
```

Viewing your Memcached metrics

This tutorial sends the following metrics to the **ECS/ContainerInsights/Prometheus** namespace in CloudWatch. You can use the CloudWatch console to see the metrics in that namespace.

Metric name	Dimensions
memcached _current_items	ClusterName , TaskDefinitionFamily
memcached _current_ connections	ClusterName , TaskDefinitionFamily
memcached _limit_bytes	ClusterName , TaskDefinitionFamily
memcached _current_bytes	ClusterName , TaskDefinitionFamily
memcached _written_ bytes_total	ClusterName , TaskDefinitionFamily
memcached _read_byt es_total	ClusterName , TaskDefinitionFamily
memcached _items_ev icted_total	ClusterName , TaskDefinitionFamily
memcached _items_re claimed_total	ClusterName , TaskDefinitionFamily
memcached _commands _total	ClusterName , TaskDefinitionFamily ClusterName , TaskDefinitionFamily, command ClusterName , TaskDefinitionFamily, status, command

> **Note**
>
> The value of the **command** dimension can be: `delete`, `get`, `cas`, `set`, `decr`, `touch`, `incr`, or `flush`.
> The value of the **status** dimension can be `hit`, `miss`, or `badval`.

You can also create a CloudWatch dashboard for your Memcached Prometheus metrics.

To create a dashboard for Memcached Prometheus metrics

1. Create environment variables, replacing the values below to match your deployment.

```
DASHBOARD_NAME=your_memcached_cw_dashboard_name
ECS_TASK_DEF_FAMILY=memcached-prometheus-demo-$ECS_CLUSTER_NAME-EC2-
$MEMCACHED_ECS_NETWORK_MOD
```

2. Enter the following command to create the dashboard.

```
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-
container-insights/latest/ecs-task-definition-templates/deployment-mode/
replica-service/cwagent-prometheus/sample_cloudwatch_dashboards/memcached/
cw_dashboard_memcached.json \
| sed "s/{{YOUR_AWS_REGION}}/$AWS_REGION/g" \
| sed "s/{{YOUR_CLUSTER_NAME}}/$ECS_CLUSTER_NAME/g" \
| sed "s/{{YOUR_TASK_DEF_FAMILY}}/$ECS_TASK_DEF_FAMILY/g" \
| xargs -0 aws cloudwatch put-dashboard --dashboard-name ${DASHBOARD_NAME} --region
 $AWS_REGION --dashboard-body
```

Tutorial for scraping Redis OSS Prometheus metrics on Amazon ECS Fargate

This tutorial provides a hands-on introduction to scrape the Prometheus metrics of a sample Redis OSS application in an Amazon ECS Fargate cluster. The Redis OSS Prometheus exporter target will be auto-discovered by the CloudWatch agent with Prometheus metric support based on the container's docker labels.

Redis OSS (https://redis.io/) is an open source (BSD licensed), in-memory data structure store, used as a database, cache and message broker. For more information, see redis.

redis_exporter (MIT License licensed) is used to expose the Redis OSS prometheus metrics on the specified port (default: 0.0.0.0:9121). For more information, see [redis_exporter].

The Docker images in the following two Docker Hub repositories are used in this tutorial:

- redis
- redis_exporter

Prerequisite

To collect metrics from a sample Prometheus workload for Amazon ECS, you must be running Container Insights in the cluster. For information about installing Container Insights, see Setting up Container Insights on Amazon ECS.

Topics

- Set the Amazon ECS Fargate cluster environment variable
- Set the network environment variables for the Amazon ECS Fargate cluster
- Install the sample Redis OSS workload
- Configure the CloudWatch agent to scrape Redis OSS Prometheus metrics
- Viewing your Redis OSS metrics

Set the Amazon ECS Fargate cluster environment variable

To set the Amazon ECS Fargate cluster environment variable

1. Install the Amazon ECS CLI if you haven't already done so. For more information, see Installing the Amazon ECS CLI.
2. Set the new Amazon ECS cluster name and Region. For example:

```
ECS_CLUSTER_NAME=ecs-fargate-redis-tutorial
AWS_DEFAULT_REGION=ca-central-1
```

3. (Optional) If you don't already have an Amazon ECS Fargate cluster where you want to install the sample Redis OSS workload and CloudWatch agent, you can create one by entering the following command.

```
ecs-cli up --capability-iam \
```

```
--cluster $ECS_CLUSTER_NAME \
--launch-type FARGATE \
--region $AWS_DEFAULT_REGION
```

The expected result of this command is as follows:

```
INFO[0000] Created cluster   cluster=ecs-fargate-redis-tutorial region=ca-central-1
INFO[0001] Waiting for your cluster resources to be created...
INFO[0001] Cloudformation stack status   stackStatus=CREATE_IN_PROGRESS
VPC created: vpc-xxxxxxxxxxxxxxxxx
Subnet created: subnet-xxxxxxxxxxxxxxxxx
Subnet created: subnet-xxxxxxxxxxxxxxxxx
Cluster creation succeeded.
```

Set the network environment variables for the Amazon ECS Fargate cluster

To set the network environment variables for the Amazon ECS Fargate cluster

1. Set your VPC and subnet ID of the Amazon ECS cluster. If you created a new cluster in the previous procedure, you'll see these values in the result of the final command. Otherwise, use the IDs of the existing cluster that you are going to use with Redis.

```
ECS_CLUSTER_VPC=vpc-xxxxxxxxxxxxxxxxx
ECS_CLUSTER_SUBNET_1=subnet-xxxxxxxxxxxxxxxxx
ECS_CLUSTER_SUBNET_2=subnet-xxxxxxxxxxxxxxxxx
```

2. In this tutorial, we are going to install the Redis OSS application and the CloudWatch agent in the default security group of the Amazon ECS cluster's VPC. The default security group allows all network connection within the same security group so the CloudWatch agent can scrape the Prometheus metrics exposed on the Redis OSS containers. In a real production environment, you might want to create dedicated security groups for the Redis OSS application and CloudWatch agent and set customized permissions for them.

Enter the following command to get the default security group ID.

```
aws ec2 describe-security-groups \
--filters Name=vpc-id,Values=$ECS_CLUSTER_VPC  \
--region $AWS_DEFAULT_REGION
```

Then set the Fargate cluster deafult security group variable by entering the following command, replacing *my-default-security-group* with the value you found from the previous command.

```
ECS_CLUSTER_SECURITY_GROUP=my-default-security-group
```

Install the sample Redis OSS workload

To install the sample Redis OSS workload which exposes the Prometheus metrics

1. Download the Redis OSS Amazon CloudFormation template by entering the following command.

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/ecs-task-definition-templates/deployment-mode/replica-service/cwagent-prometheus/sample_traffic/redis/redis-traffic-sample.yaml
```

2. Set the IAM role names to be created for Redis OSS by entering the following commands.

```
REDIS_ECS_TASK_ROLE_NAME=redis-prometheus-demo-ecs-task-role-name
REDIS_ECS_EXECUTION_ROLE_NAME=redis-prometheus-demo-ecs-execution-role-name
```

3. Install the sample Redis OSS workload by entering the following command.

```
aws cloudformation create-stack --stack-name Redis-Prometheus-Demo-ECS-$ECS_CLUSTER_NAME-fargate-awsvpc \
    --template-body file://redis-traffic-sample.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=$ECS_CLUSTER_NAME \
                 ParameterKey=SecurityGroupID,ParameterValue=$ECS_CLUSTER_SECURITY_GROUP \
                 ParameterKey=SubnetID,ParameterValue=$ECS_CLUSTER_SUBNET_1 \
                 ParameterKey=TaskRoleName,ParameterValue=$REDIS_ECS_TASK_ROLE_NAME \
                 ParameterKey=ExecutionRoleName,ParameterValue=$REDIS_ECS_EXECUTION_ROLE_NAME \
    --capabilities CAPABILITY_NAMED_IAM \
    --region $AWS_DEFAULT_REGION
```

The Amazon CloudFormation stack creates four resources:

- One ECS task role
- One ECS task execution role
- One Redis OSS task definition
- One Redis OSS service

In the Redis OSS task definition, two containers are defined:

- The primary container runs a simple Redis OSS application and opens port 6379 for access.
- The other container runs the Redis OSS exporter process to expose the Prometheus metrics on port 9121. This is the container to be discovered and scraped by the CloudWatch agent. The following docker label is defined so that the CloudWatch agent can discover this container based on it.

```
ECS_PROMETHEUS_EXPORTER_PORT: 9121
```

Configure the CloudWatch agent to scrape Redis OSS Prometheus metrics

To configure the CloudWatch agent to scrape Redis OSS Prometheus metrics

1. Download the latest version of `cwagent-ecs-prometheus-metric-for-awsvpc.yaml` by entering the following command.

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/ecs-task-definition-templates/deployment-mode/replica-service/cwagent-prometheus/cloudformation-quickstart/cwagent-ecs-prometheus-metric-for-awsvpc.yaml
```

2. Open the file with a text editor, and find the full CloudWatch agent configuration behind the `value` key in the `resource:CWAgentConfigSSMParameter` section.

   Then, in the `ecs_service_discovery` section shown here, the `docker_label`-based service discovery is enabled with the default settings which are based on `ECS_PROMETHEUS_EXPORTER_PORT`, which matches the docker label we defined in the Redis OSS ECS task definition. So we do not need to make any changes in this section:

```
ecs_service_discovery": {
  "sd_frequency": "1m",
```

```
  "sd_result_file": "/tmp/cwagent_ecs_auto_sd.yaml",
*  "docker_label": {
  },*
  ...
```

For the `metric_declaration` section, the default setting does not allow any Redis OSS metrics. Add the following section to allow Redis OSS metrics. Be sure to follow the existing indentation pattern.

```
{
  "source_labels": ["container_name"],
  "label_matcher": "^redis-exporter-.*$",
  "dimensions": [["ClusterName","TaskDefinitionFamily"]],
  "metric_selectors": [
    "^redis_net_(in|out)put_bytes_total$",
    "^redis_(expired|evicted)_keys_total$",
    "^redis_keyspace_(hits|misses)_total$",
    "^redis_memory_used_bytes$",
    "^redis_connected_clients$"
  ]
},
{
  "source_labels": ["container_name"],
  "label_matcher": "^redis-exporter-.*$",
  "dimensions": [["ClusterName","TaskDefinitionFamily","cmd"]],
  "metric_selectors": [
    "^redis_commands_total$"
  ]
},
{
  "source_labels": ["container_name"],
  "label_matcher": "^redis-exporter-.*$",
  "dimensions": [["ClusterName","TaskDefinitionFamily","db"]],
  "metric_selectors": [
    "^redis_db_keys$"
  ]
},
```

3. If you already have the CloudWatch agent deployed in the Amazon ECS cluster by Amazon CloudFormation, you can create a change set by entering the following commands.

```
ECS_LAUNCH_TYPE=FARGATE
```

```
CREATE_IAM_ROLES=True
ECS_CLUSTER_SUBNET=$ECS_CLUSTER_SUBNET_1
ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name

aws cloudformation create-change-set --stack-name CWAgent-Prometheus-ECS-
$ECS_CLUSTER_NAME-$ECS_LAUNCH_TYPE-awsvpc \
    --template-body file://cwagent-ecs-prometheus-metric-for-awsvpc.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=$ECS_CLUSTER_NAME \
                 ParameterKey=CreateIAMRoles,ParameterValue=$CREATE_IAM_ROLES \
                 ParameterKey=ECSLaunchType,ParameterValue=$ECS_LAUNCH_TYPE \
                 ParameterKey=SecurityGroupID,ParameterValue=
$ECS_CLUSTER_SECURITY_GROUP \
                 ParameterKey=SubnetID,ParameterValue=$ECS_CLUSTER_SUBNET \
                 ParameterKey=TaskRoleName,ParameterValue=$ECS_TASK_ROLE_NAME \
                 ParameterKey=ExecutionRoleName,ParameterValue=
$ECS_EXECUTION_ROLE_NAME \
    --capabilities CAPABILITY_NAMED_IAM \
    --region ${AWS_DEFAULT_REGION} \
    --change-set-name redis-scraping-support
```

4. Open the Amazon CloudFormation console at https://console.amazonaws.cn/cloudformation.
5. Review the newly created changeset `redis-scraping-support`. You should see one change applied to the `CWAgentConfigSSMParameter` resource. Execute the changeset and restart the CloudWatch agent task by entering the following commands.

```
aws ecs update-service --cluster $ECS_CLUSTER_NAME \
--desired-count 0 \
--service cwagent-prometheus-replica-service-$ECS_LAUNCH_TYPE-awsvpc \
--region ${AWS_DEFAULT_REGION}
```

6. Wait about 10 seconds, and then enter the following command.

```
aws ecs update-service --cluster $ECS_CLUSTER_NAME \
--desired-count 1 \
--service cwagent-prometheus-replica-service-$ECS_LAUNCH_TYPE-awsvpc \
--region ${AWS_DEFAULT_REGION}
```

7. If you are installing the CloudWatch agent with Prometheus metric collecting for the cluster for the first time, please enter the following commands:

```
ECS_LAUNCH_TYPE=FARGATE
```

```
CREATE_IAM_ROLES=True
ECS_CLUSTER_SUBNET=$ECS_CLUSTER_SUBNET_1
ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name

aws cloudformation create-stack --stack-name CWAgent-Prometheus-ECS-
$ECS_CLUSTER_NAME-$ECS_LAUNCH_TYPE-awsvpc \
    --template-body file://cwagent-ecs-prometheus-metric-for-awsvpc.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=$ECS_CLUSTER_NAME \
                 ParameterKey=CreateIAMRoles,ParameterValue=$CREATE_IAM_ROLES \
                 ParameterKey=ECSLaunchType,ParameterValue=$ECS_LAUNCH_TYPE \
                 ParameterKey=SecurityGroupID,ParameterValue=
$ECS_CLUSTER_SECURITY_GROUP \
                 ParameterKey=SubnetID,ParameterValue=$ECS_CLUSTER_SUBNET \
                 ParameterKey=TaskRoleName,ParameterValue=$ECS_TASK_ROLE_NAME \
                 ParameterKey=ExecutionRoleName,ParameterValue=
$ECS_EXECUTION_ROLE_NAME \
    --capabilities CAPABILITY_NAMED_IAM \
    --region ${AWS_DEFAULT_REGION}
```

Viewing your Redis OSS metrics

This tutorial sends the following metrics to the **ECS/ContainerInsights/Prometheus** namespace in CloudWatch. You can use the CloudWatch console to see the metrics in that namespace.

Metric Name	Dimensions	
`redis_net_input_bytes_total`	ClusterName, `TaskDefinitionFamily`	
`redis_net_output_bytes_total`	ClusterName, `TaskDefinitionFamily`	
`redis_expired_keys_total`	ClusterName, `TaskDefinitionFamily`	

Metric Name	Dimensions
redis_evicted_keys_total	ClusterName, TaskDefinitionFamily
redis_keyspace_hits_total	ClusterName, TaskDefinitionFamily
redis_keyspace_misses_total	ClusterName, TaskDefinitionFamily
redis_memory_used_bytes	ClusterName, TaskDefinitionFamily
redis_connected_clients	ClusterName, TaskDefinitionFamily
redis_commands_total	ClusterName , TaskDefinitionFamily , cmd
redis_db_keys	ClusterName , TaskDefinitionFamily , db

Note

The value of the **cmd** dimension can be: append, client, command, config, dbsize, flushall, get, incr, info, latency, or slowlog.
The value of the **db** dimension can be db0 to db15.

You can also create a CloudWatch dashboard for your Redis OSS Prometheus metrics.

To create a dashboard for Redis OSS Prometheus metrics

1. Create environment variables, replacing the values below to match your deployment.

```
DASHBOARD_NAME=your_cw_dashboard_name
ECS_TASK_DEF_FAMILY=redis-prometheus-demo-$ECS_CLUSTER_NAME-fargate-awsvpc
```

2. Enter the following command to create the dashboard.

```
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-
prometheus/sample_cloudwatch_dashboards/redis/cw_dashboard_redis.json \
| sed "s/{{YOUR_AWS_REGION}}/${REGION_NAME}/g" \
| sed "s/{{YOUR_CLUSTER_NAME}}/${CLUSTER_NAME}/g" \
| sed "s/{{YOUR_NAMESPACE}}/${NAMESPACE}/g" \
```

Set up and configure Prometheus metrics collection on Amazon EKS and Kubernetes clusters

To collect Prometheus metrics from clusters running Amazon EKS or Kubernetes, you can use the CloudWatch agent as a collector or use the Amazon Distro for OpenTelemetry collector. For information about using the Amazon Distro for OpenTelemetry collector, see https://aws-otel.github.io/docs/getting-started/container-insights/eks-prometheus.

The following sections explain how to collect Prometheus metrics using the CloudWatch agent. They explain how to install the CloudWatch agent with Prometheus monitoring on clusters running Amazon EKS or Kubernetes, and how to configure the agent to scrape additional targets. They also provide optional tutorials for setting up sample workloads to use for testing with Prometheus monitoring.

Topics

Install the CloudWatch agent with Prometheus metrics collection on Amazon EKS and Kubernetes clusters

This section explains how to set up the CloudWatch agent with Prometheus monitoring in a cluster running Amazon EKS or Kubernetes. After you do this, the agent automatically scrapes and imports metrics for the following workloads running in that cluster.

- Amazon App Mesh

- NGINX
- Memcached
- Java/JMX
- HAProxy
- Fluent Bit

You can also configure the agent to scrape and import additional Prometheus workloads and sources.

Before following these steps to install the CloudWatch agent for Prometheus metric collection, you must have a cluster running on Amazon EKS or a Kubernetes cluster running on an Amazon EC2 instance.

VPC security group requirements

The ingress rules of the security groups for the Prometheus workloads must open the Prometheus ports to the CloudWatch agent for scraping the Prometheus metrics by the private IP.

The egress rules of the security group for the CloudWatch agent must allow the CloudWatch agent to connect to the Prometheus workloads' port by private IP.

Topics

Install the CloudWatch agent with Prometheus metrics collection on Amazon EKS and Kubernetes clusters

This section explains how to set up the CloudWatch agent with Prometheus monitoring in a cluster running Amazon EKS or Kubernetes. After you do this, the agent automatically scrapes and imports metrics for the following workloads running in that cluster.

- Amazon App Mesh
- NGINX
- Memcached

- Java/JMX
- HAProxy
- Fluent Bit

You can also configure the agent to scrape and import additional Prometheus workloads and sources.

Before following these steps to install the CloudWatch agent for Prometheus metric collection, you must have a cluster running on Amazon EKS or a Kubernetes cluster running on an Amazon EC2 instance.

VPC security group requirements

The ingress rules of the security groups for the Prometheus workloads must open the Prometheus ports to the CloudWatch agent for scraping the Prometheus metrics by the private IP.

The egress rules of the security group for the CloudWatch agent must allow the CloudWatch agent to connect to the Prometheus workloads' port by private IP.

Topics

Setting up IAM roles

The first step is to set up the necessary IAM role in the cluster. There are two methods:

- Set up an IAM role for a service account, also known as a *service role*. This method works for both the EC2 launch type and the Fargate launch type.
- Add an IAM policy to the IAM role used for the cluster. This works only for the EC2 launch type.

Set up a service role (EC2 launch type and Fargate launch type)

To set up a service role, enter the following command. Replace *MyCluster* with the name of the cluster.

```
eksctl create iamserviceaccount \
 --name cwagent-prometheus \
```

```
--namespace amazon-cloudwatch \
 --cluster MyCluster \
--attach-policy-arn arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy \
--approve \
--override-existing-serviceaccounts
```

Add a policy to the node group's IAM role (EC2 launch type only)

To set up the IAM policy in a node group for Prometheus support

1. Open the Amazon EC2 console at https://console.amazonaws.cn/ec2/.
2. In the navigation pane, choose **Instances**.
3. You need to find the prefix of the IAM role name for the cluster. To do this, select the check box next to the name of an instance that is in the cluster, and choose **Actions**, **Security**, **Modify IAM Role**. Then copy the prefix of the IAM role, such as `eksctl-dev303-workshop-nodegroup`.
4. Open the IAM console at https://console.amazonaws.cn/iam/.
5. In the navigation pane, choose **Roles**.
6. Use the search box to find the prefix that you copied earlier in this procedure, and choose that role.
7. Choose **Attach policies**.
8. Use the search box to find **CloudWatchAgentServerPolicy**. Select the check box next to **CloudWatchAgentServerPolicy**, and choose **Attach policy**.

Installing the CloudWatch agent to collect Prometheus metrics

You must install the CloudWatch agent in the cluster to collect the metrics. How to install the agent differs for Amazon EKS clusters and Kubernetes clusters.

Delete previous versions of the CloudWatch agent with Prometheus support

If you have already installed a version of the CloudWatch agent with Prometheus support in your cluster, you must delete that version by entering the following command. This is necessary only for previous versions of the agent with Prometheus support. You do not need to delete the CloudWatch agent that enables Container Insights without Prometheus support.

```
kubectl delete deployment cwagent-prometheus -n amazon-cloudwatch
```

Installing the CloudWatch agent on Amazon EKS clusters with the EC2 launch type

To install the CloudWatch agent with Prometheus support on an Amazon EKS cluster, follow these steps.

To install the CloudWatch agent with Prometheus support on an Amazon EKS cluster

1. Enter the following command to check whether the `amazon-cloudwatch` namespace has already been created:

   ```
   kubectl get namespace
   ```

2. If `amazon-cloudwatch` is not displayed in the results, create it by entering the following command:

   ```
   kubectl create namespace amazon-cloudwatch
   ```

3. To deploy the agent with the default configuration and have it send data to the Amazon Region that it is installed in, enter the following command:

   ```
   kubectl apply -f https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-prometheus/prometheus-eks.yaml
   ```

   To have the agent send data to a different Region instead, follow these steps:

   a. Download the YAML file for the agent by entering the following command:

      ```
      curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-prometheus/prometheus-eks.yaml
      ```

   b. Open the file with a text editor, and search for the `cwagentconfig.json` block of the file.

   c. Add the highlighted lines, specifying the Region that you want:

      ```
      cwagentconfig.json: |
          {
            "agent": {
              "region": "us-east-2"
            },
      ```

```
    "logs": { ...
```

d. Save the file and deploy the agent using your updated file.

```
kubectl apply -f prometheus-eks.yaml
```

Installing the CloudWatch agent on Amazon EKS clusters with the Fargate launch type

To install the CloudWatch agent with Prometheus support on an Amazon EKS cluster with the Fargate launch type, follow these steps.

To install the CloudWatch agent with Prometheus support on an Amazon EKS cluster with the Fargate launch type

1. Enter the following command to create a Fargate profile for the CloudWatch agent so that it can run inside the cluster. Replace *MyCluster* with the name of the cluster.

```
eksctl create fargateprofile --cluster MyCluster \
--name amazon-cloudwatch \
--namespace amazon-cloudwatch
```

2. To install the CloudWatch agent, enter the following command. Replace *MyCluster* with the name of the cluster. This name is used in the log group name that stores the log events collected by the agent, and is also used as a dimension for the metrics collected by the agent.

   Replace *region* with the name of the Region where you want the metrics to be sent. For example, `us-west-1`.

```
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-prometheus/prometheus-eks-fargate.yaml | 
sed "s/{{cluster_name}}/MyCluster/;s/{{region_name}}/region/" | 
kubectl apply -f -
```

Installing the CloudWatch agent on a Kubernetes cluster

To install the CloudWatch agent with Prometheus support on a cluster running Kubernetes, enter the following command:

```
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-
prometheus/prometheus-k8s.yaml | 
sed "s/{{cluster_name}}/MyCluster/;s/{{region_name}}/region/" | 
kubectl apply -f -
```

Replace *MyCluster* with the name of the cluster. This name is used in the log group name that stores the log events collected by the agent, and is also used as a dimension for the metrics collected by the agent.

Replace *region* with the name of the Amazon Region where you want the metrics to be sent. For example, **us-west-1**.

Verify that the agent is running

On both Amazon EKS and Kubernetes clusters, you can enter the following command to confirm that the agent is running.

```
kubectl get pod -l "app=cwagent-prometheus" -n amazon-cloudwatch
```

If the results include a single CloudWatch agent pod in the `Running` state, the agent is running and collecting Prometheus metrics. By default the CloudWatch agent collects metrics for App Mesh, NGINX, Memcached, Java/JMX, and HAProxy every minute. For more information about those metrics, see Prometheus metrics collected by the CloudWatch agent. For instructions on how to see your Prometheus metrics in CloudWatch, see Viewing your Prometheus metrics

You can also configure the CloudWatch agent to collect metrics from other Prometheus exporters. For more information, see Scraping additional Prometheus sources and importing those metrics.

Scraping additional Prometheus sources and importing those metrics

The CloudWatch agent with Prometheus monitoring needs two configurations to scrape the Prometheus metrics. One is for the standard Prometheus configurations as documented in <scrape_config> in the Prometheus documentation. The other is for the CloudWatch agent configuration.

For Amazon EKS clusters, the configurations are defined in `prometheus-eks.yaml` (for the EC2 launch type) or `prometheus-eks-fargate.yaml` (for the Fargate launch type) as two config maps:

- The `name: prometheus-config` section contains the settings for Prometheus scraping.
- The `name: prometheus-cwagentconfig` section contains the configuration for the CloudWatch agent. You can use this section to configure how the Prometheus metrics are collected by CloudWatch. For example, you specify which metrics are to be imported into CloudWatch, and define their dimensions.

For Kubernetes clusters running on Amazon EC2 instances, the configurations are defined in the `prometheus-k8s.yaml` YAML file as two config maps:

- The `name: prometheus-config` section contains the settings for Prometheus scraping.
- The `name: prometheus-cwagentconfig` section contains the configuration for the CloudWatch agent.

To scrape additional Prometheus metrics sources and import those metrics to CloudWatch, you modify both the Prometheus scrape configuration and the CloudWatch agent configuration, and then re-deploy the agent with the updated configuration.

VPC security group requirements

The ingress rules of the security groups for the Prometheus workloads must open the Prometheus ports to the CloudWatch agent for scraping the Prometheus metrics by the private IP.

The egress rules of the security group for the CloudWatch agent must allow the CloudWatch agent to connect to the Prometheus workloads' port by private IP.

Prometheus scrape configuration

The CloudWatch agent supports the standard Prometheus scrape configurations as documented in <scrape_config> in the Prometheus documentation. You can edit this section to update the configurations that are already in this file, and add additional Prometheus scraping targets. By default, the sample configuration file contains the following global configuration lines:

```
global:
  scrape_interval: 1m
  scrape_timeout: 10s
```

- **scrape_interval**— Defines how frequently to scrape targets.
- **scrape_timeout**— Defines how long to wait before a scrape request times out.

You can also define different values for these settings at the job level, to override the global configurations.

Prometheus scraping jobs

The CloudWatch agent YAML files already have some default scraping jobs configured. For example, in `prometheus-eks.yaml`, the default scraping jobs are configured in the `job_name` lines in the `scrape_configs` section. In this file, the following default `kubernetes-pod-jmx` section scrapes JMX exporter metrics.

```
  - job_name: 'kubernetes-pod-jmx'
     sample_limit: 10000
     metrics_path: /metrics
     kubernetes_sd_configs:
     - role: pod
     relabel_configs:
     - source_labels: [__address__]
       action: keep
       regex: '.*:9404$'
     - action: labelmap
       regex: __meta_kubernetes_pod_label_(.+)
     - action: replace
       source_labels:
       - __meta_kubernetes_namespace
       target_label: Namespace
     - source_labels: [__meta_kubernetes_pod_name]
       action: replace
       target_label: pod_name
     - action: replace
       source_labels:
       - __meta_kubernetes_pod_container_name
       target_label: container_name
     - action: replace
       source_labels:
       - __meta_kubernetes_pod_controller_name
       target_label: pod_controller_name
     - action: replace
       source_labels:
       - __meta_kubernetes_pod_controller_kind
       target_label: pod_controller_kind
     - action: replace
       source_labels:
       - __meta_kubernetes_pod_phase
```

```
        target_label: pod_phase
```

Each of these default targets are scraped, and the metrics are sent to CloudWatch in log events using embedded metric format. For more information, see Embedding metrics within logs.

Log events from Amazon EKS and Kubernetes clusters are stored in the **/aws/containerinsights/*cluster_name*/prometheus** log group in CloudWatch Logs. Log events from Amazon ECS clusters are stored in the **/aws/ecs/containerinsights/*cluster_name*/prometheus** log group.

Each scraping job is contained in a different log stream in this log group. For example, the Prometheus scraping job `kubernetes-pod-appmesh-envoy` is defined for App Mesh. All App Mesh Prometheus metrics from Amazon EKS and Kubernetes clusters are sent to the log stream named **/aws/containerinsights/*cluster_name*>prometheus/kubernetes-pod-appmesh-envoy/**.

To add a new scraping target, you add a new `job_name` section to the `scrape_configs` section of the YAML file, and restart the agent. For an example of this process, see Tutorial for adding a new Prometheus scrape target: Prometheus API Server metrics.

CloudWatch agent configuration for Prometheus

The CloudWatch agent configuration file has a `prometheus` section under `metrics_collected` for the Prometheus scraping configuration. It includes the following configuration options:

- **cluster_name**— specifies the cluster name to be added as a label in the log event. This field is optional. If you omit it, the agent can detect the Amazon EKS or Kubernetes cluster name.
- **log_group_name**— specifies the log group name for the scraped Prometheus metrics. This field is optional. If you omit it, CloudWatch uses **/aws/containerinsights/*cluster_name*/prometheus** for logs from Amazon EKS and Kubernetes clusters.
- **prometheus_config_path**— specifies the Prometheus scrape configuration file path. If the value of this field starts with `env:` the Prometheus scrape configuration file contents will be retrieved from the container's environment variable. Do not change this field.
- **ecs_service_discovery**— is the section to specify the configuration for Amazon ECS Prometheus service discovery. For more information, see Detailed guide for autodiscovery on Amazon ECS clusters.

  The `ecs_service_discovery` section can contain the following fields:

- `sd_frequency` is the frequency to discover the Prometheus exporters. Specify a number and a unit suffix. For example, `1m` for once per minute or `30s` for once per 30 seconds. Valid unit suffixes are `ns`, `us`, `ms`, `s`, `m`, and `h`.

  This field is optional. The default is 60 seconds (1 minute).
- `sd_target_cluster` is the target Amazon ECS cluster name for auto-discovery. This field is optional. The default is the name of the Amazon ECS cluster where the CloudWatch agent is installed.
- `sd_cluster_region` is the target Amazon ECS cluster's Region. This field is optional. The default is the Region of the Amazon ECS cluster where the CloudWatch agent is installed. .
- `sd_result_file` is the path of the YAML file for the Prometheus target results. The Prometheus scrape configuration will refer to this file.
- `docker_label` is an optional section that you can use to specify the configuration for docker label-based service discovery. If you omit this section, docker label-based discovery is not used. This section can contain the following fields:
  - `sd_port_label` is the container's docker label name that specifies the container port for Prometheus metrics. The default value is `ECS_PROMETHEUS_EXPORTER_PORT`. If the container does not have this docker label, the CloudWatch agent will skip it.
  - `sd_metrics_path_label` is the container's docker label name that specifies the Prometheus metrics path. The default value is `ECS_PROMETHEUS_METRICS_PATH`. If the container does not have this docker label, the agent assumes the default path `/metrics`.
  - `sd_job_name_label` is the container's docker label name that specifies the Prometheus scrape job name. The default value is `job`. If the container does not have this docker label, the CloudWatch agent uses the job name in the Prometheus scrape configuration.
- `task_definition_list` is an optional section that you can use to specify the configuration of task definition-based service discovery. If you omit this section, task definition-based discovery is not used. This section can contain the following fields:
  - `sd_task_definition_arn_pattern` is the pattern to use to specify the Amazon ECS task definitions to discover. This is a regular expression.
  - `sd_metrics_ports` lists the containerPort for the Prometheus metrics. Separate the containerPorts with semicolons.
  - `sd_container_name_pattern` specifies the Amazon ECS task container names. This is a regular expression.

  - `sd_metrics_path` specifies the Prometheus metric path. If you omit this, the agent assumes the default path `/metrics`
  - `sd_job_name` specifies the Prometheus scrape job name. If you omit this field, the CloudWatch agent uses the job name in the Prometheus scrape configuration.
- **metric_declaration—** are sections that specify the array of logs with embedded metric format to be generated. There are `metric_declaration` sections for each Prometheus source that the CloudWatch agent imports from by default. These sections each include the following fields:
  - `label_matcher` is a regular expression that checks the value of the labels listed in `source_labels`. The metrics that match are enabled for inclusion in the embedded metric format sent to CloudWatch.

    If you have multiple labels specified in `source_labels`, we recommend that you do not use ^ or $ characters in the regular expression for `label_matcher`.
  - `source_labels` specifies the value of the labels that are checked by the `label_matcher` line.
  - `label_separator` specifies the separator to be used in the `label_matcher` line if multiple `source_labels` are specified. The default is `;`. You can see this default used in the `label_matcher` line in the following example.
  - `metric_selectors` is a regular expression that specifies the metrics to be collected and sent to CloudWatch.
  - `dimensions` is the list of labels to be used as CloudWatch dimensions for each selected metric.

See the following `metric_declaration` example.

```
"metric_declaration": [
  {
     "source_labels":[ "Service", "Namespace"],
     "label_matcher":"(.*node-exporter.*|.*kube-dns.*);kube-system",
     "dimensions":[
        ["Service", "Namespace"]
     ],
     "metric_selectors":[
        "^coredns_dns_request_type_count_total$"
     ]
  }
```

```
]
```

This example configures an embedded metric format section to be sent as a log event if the following conditions are met:

- The value of `Service` contains either `node-exporter` or `kube-dns`.
- The value of `Namespace` is `kube-system`.
- The Prometheus metric `coredns_dns_request_type_count_total` contains both `Service` and `Namespace` labels.

The log event that is sent includes the following highlighted section:

```
{
   "CloudWatchMetrics":[
      {
         "Metrics":[
            {
               "Name":"coredns_dns_request_type_count_total"
            }
         ],
         "Dimensions":[
            [
               "Namespace",
               "Service"
            ]
         ],
         "Namespace":"ContainerInsights/Prometheus"
      }
   ],
   "Namespace":"kube-system",
   "Service":"kube-dns",
   "coredns_dns_request_type_count_total":2562,
   "eks_amazonaws_com_component":"kube-dns",
   "instance":"192.168.61.254:9153",
   "job":"kubernetes-service-endpoints",
   ...
}
```

Tutorial for adding a new Prometheus scrape target: Prometheus API Server metrics

The Kubernetes API Server exposes Prometheus metrics on endpoints by default. The official example for the Kubernetes API Server scraping configuration is available on Github.

The following tutorial shows how to do the following steps to begin importing Kubernetes API Server metrics into CloudWatch:

- Adding the Prometheus scraping configuration for Kubernetes API Server to the CloudWatch agent YAML file.
- Configuring the embedded metric format metrics definitions in the CloudWatch agent YAML file.
- (Optional) Creating a CloudWatch dashboard for the Kubernetes API Server metrics.

> **Note**
>
> The Kubernetes API Server exposes gauge, counter, histogram, and summary metrics. In this release of Prometheus metrics support, CloudWatch imports only the metrics with gauge, counter, and summary types.

To start collecting Kubernetes API Server Prometheus metrics in CloudWatch

1. Download the latest version of the `prometheus-eks.yaml`, `prometheus-eks-fargate.yaml`, or `prometheus-k8s.yaml` file by entering one of the following commands.

   For an Amazon EKS cluster with the EC2 launch type, enter the following command:

   ```
   curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-prometheus/prometheus-eks.yaml
   ```

   For an Amazon EKS cluster with the Fargate launch type, enter the following command:

   ```
   curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-prometheus/prometheus-eks-fargate.yaml
   ```

   For a Kubernetes cluster running on an Amazon EC2 instance, enter the following command:

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-prometheus/prometheus-k8s.yaml
```

2. Open the file with a text editor, find the `prometheus-config` section, and add the following section inside of that section. Then save the changes:

```
    # Scrape config for API servers
    - job_name: 'kubernetes-apiservers'
      kubernetes_sd_configs:
        - role: endpoints
          namespaces:
            names:
              - default
      scheme: https
      tls_config:
        ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
        insecure_skip_verify: true
      bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
      relabel_configs:
      - source_labels: [__meta_kubernetes_service_name, 
 __meta_kubernetes_endpoint_port_name]
        action: keep
        regex: kubernetes;https
      - action: replace
        source_labels:
        - __meta_kubernetes_namespace
        target_label: Namespace
      - action: replace
        source_labels:
        - __meta_kubernetes_service_name
        target_label: Service
```

3. While you still have the YAML file open in the text editor, find the `cwagentconfig.json` section. Add the following subsection and save the changes. This section puts the API server metrics onto the CloudWatch agent allow list. Three types of API Server metrics are added to the allow list:

   - etcd object counts
   - API Server registration controller metrics
   - API Server request metrics

```
{"source_labels": ["job", "resource"],
  "label_matcher": "^kubernetes-apiservers;(services|daemonsets.apps|
deployments.apps|configmaps|endpoints|secrets|serviceaccounts|replicasets.apps)",
  "dimensions": [["ClusterName","Service","resource"]],
  "metric_selectors": [
  "^etcd_object_counts$"
  ]
},
{"source_labels": ["job", "name"],
   "label_matcher": "^kubernetes-apiservers;APIServiceRegistrationController$",
   "dimensions": [["ClusterName","Service","name"]],
   "metric_selectors": [
   "^workqueue_depth$",
   "^workqueue_adds_total$",
   "^workqueue_retries_total$"
  ]
},
{"source_labels": ["job","code"],
  "label_matcher": "^kubernetes-apiservers;2[0-9]{2}$",
  "dimensions": [["ClusterName","Service","code"]],
  "metric_selectors": [
   "^apiserver_request_total$"
  ]
},
{"source_labels": ["job"],
  "label_matcher": "^kubernetes-apiservers",
  "dimensions": [["ClusterName","Service"]],
  "metric_selectors": [
  "^apiserver_request_total$"
  ]
},
```

4. If you already have the CloudWatch agent with Prometheus support deployed in the cluster, you must delete it by entering the following command:

```
kubectl delete deployment cwagent-prometheus -n amazon-cloudwatch
```

5. Deploy the CloudWatch agent with your updated configuration by entering one of the following commands. For an Amazon EKS cluster with the EC2 launch type, enter:

```
kubectl apply -f prometheus-eks.yaml
```

For an Amazon EKS cluster with the Fargate launch type, enter the following command. Replace *MyCluster* and *region* with values to match your deployment.

```
cat prometheus-eks-fargate.yaml \
| sed "s/{{cluster_name}}/MyCluster/;s/{{region_name}}/region/" \
| kubectl apply -f -
```

For a Kubernetes cluster, enter the following command. Replace *MyCluster* and *region* with values to match your deployment.

```
cat prometheus-k8s.yaml \
| sed "s/{{cluster_name}}/MyCluster/;s/{{region_name}}/region/" \
| kubectl apply -f -
```

Once you have done this, you should see a new log stream named **kubernetes-apiservers** in the **/aws/containerinsights/*cluster_name*/prometheus** log group. This log stream should include log events with an embedded metric format definition like the following:

```
{
   "CloudWatchMetrics":[
      {
         "Metrics":[
            {
               "Name":"apiserver_request_total"
            }
         ],
         "Dimensions":[
            [
               "ClusterName",
               "Service"
            ]
         ],
         "Namespace":"ContainerInsights/Prometheus"
      }
   ],
   "ClusterName":"my-cluster-name",
   "Namespace":"default",
```

```
    "Service":"kubernetes",
    "Timestamp":"1592267020339",
    "Version":"0",
    "apiserver_request_count":0,
    "apiserver_request_total":0,
    "code":"0",
    "component":"apiserver",
    "contentType":"application/json",
    "instance":"192.0.2.0:443",
    "job":"kubernetes-apiservers",
    "prom_metric_type":"counter",
    "resource":"pods",
    "scope":"namespace",
    "verb":"WATCH",
    "version":"v1"
}
```


You can view your metrics in the CloudWatch console in the **ContainerInsights/Prometheus** namespace. You can also optionally create a CloudWatch dashboard for your Prometheus Kubernetes API Server metrics.

(Optional) Creating a dashboard for Kubernetes API Server metrics

To see Kubernetes API Server metrics in your dashboard, you must have first completed the steps in the previous sections to start collecting these metrics in CloudWatch.

To create a dashboard for Kubernetes API Server metrics

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. Make sure you have the correct Amazon Region selected.
3. In the navigation pane, choose **Dashboards**.
4. Choose **Create Dashboard**. Enter a name for the new dashboard, and choose **Create dashboard**.
5. In **Add to this dashboard**, choose **Cancel**.
6. Choose **Actions**, **View/edit source**.
7. Download the following JSON file: Kubernetes API Dashboard source.
8. Open the JSON file that you downloaded with a text editor, and make the following changes:

   - Replace all the `{{YOUR_CLUSTER_NAME}}` strings with the exact name of your cluster. Make sure not to add whitespaces before or after the text.

- Replace all the `{{YOUR_AWS_REGION}}` strings with the name of the Region where the metrics are collected. For example `us-west-2`. Be sure not to add whitespaces before or after the text.

9. Copy the entire JSON blob and paste it into the text box in the CloudWatch console, replacing what is already in the box.
10. Choose **Update**, **Save dashboard**.

(Optional) Set up sample containerized Amazon EKS workloads for Prometheus metric testing

To test the Prometheus metric support in CloudWatch Container Insights, you can set up one or more of the following containerized workloads. The CloudWatch agent with Prometheus support automatically collects metrics from each of these workloads. To see the metrics that are collected by default, see Prometheus metrics collected by the CloudWatch agent.

Before you can install any of these workloads, you must install Helm 3.x by entering the following commands:

```
brew install helm
```

For more information, see Helm.

Topics

- Set up Amazon App Mesh sample workload for Amazon EKS and Kubernetes
- Set up NGINX with sample traffic on Amazon EKS and Kubernetes
- Set up memcached with a metric exporter on Amazon EKS and Kubernetes
- Set up Java/JMX sample workload on Amazon EKS and Kubernetes
- Set up HAProxy with a metric exporter on Amazon EKS and Kubernetes
- Tutorial for adding a new Prometheus scrape target: Redis OSS on Amazon EKS and Kubernetes clusters

Set up Amazon App Mesh sample workload for Amazon EKS and Kubernetes

Prometheus support in CloudWatch Container Insights supports Amazon App Mesh. The following sections explain how to set up App Mesh.

Topics

- [Set up Amazon App Mesh sample workload on an Amazon EKS cluster with the EC2 launch type or a Kubernetes cluster]
- [Set up Amazon App Mesh sample workload on an Amazon EKS cluster with the Fargate launch type]

Set up Amazon App Mesh sample workload on an Amazon EKS cluster with the EC2 launch type or a Kubernetes cluster

Use these instructions if you are setting up App Mesh on a cluster running Amazon EKS with the EC2 launch type, or a Kubernetes cluster.

Configure IAM permissions

You must add the **AWSAppMeshFullAccess** policy to the IAM role for your Amazon EKS or Kubernetes node group. On Amazon EKS, this node group name looks similar to `eksctl-integ-test-eks-prometheus-NodeInstanceRole-ABCDEFHIJKL`. On Kubernetes, it might look similar to `nodes.integ-test-kops-prometheus.k8s.local`.

Install App Mesh

To install the App Mesh Kubernetes controller, follow the instructions in App Mesh Controller.

Install a sample application

aws-app-mesh-examples contains several Kubernetes App Mesh walkthroughs. For this tutorial, you install a sample color application that shows how http routes can use headers for matching incoming requests.

To use a sample App Mesh application to test Container Insights

1. Install the application using these instructions: https://github.com/aws/aws-app-mesh-examples/tree/main/walkthroughs/howto-k8s-http-headers.
2. Launch a curler pod to generate traffic:

```
kubectl -n default run -it curler --image=tutum/curl /bin/bash
```

3. Curl different endpoints by changing HTTP headers. Run the curl command multiple times, as shown:

```
curl -H "color_header: blue" front.howto-k8s-http-headers.svc.cluster.local:8080/;
 echo;

curl -H "color_header: red" front.howto-k8s-http-headers.svc.cluster.local:8080/;
 echo;

curl -H "color_header: yellow" front.howto-k8s-http-
headers.svc.cluster.local:8080/; echo;
```

4. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
5. In the Amazon Region where your cluster is running, choose **Metrics** in the navigation pane. The metric are in the **ContainerInsights/Prometheus** namespace.
6. To see the CloudWatch Logs events, choose **Log groups** in the navigation pane. The events are in the log group `/aws/containerinsights/your_cluster_name/prometheus` in the log stream `kubernetes-pod-appmesh-envoy`.

Deleting the App Mesh test environment

When you have finished using App Mesh and the sample application, use the following commands to delete the unnecessary resources. Delete the sample application by entering the following command:

```
cd aws-app-mesh-examples/walkthroughs/howto-k8s-http-headers/
kubectl delete -f _output/manifest.yaml
```

Delete the App Mesh controller by entering the following command:

```
helm delete appmesh-controller -n appmesh-system
```

Set up Amazon App Mesh sample workload on an Amazon EKS cluster with the Fargate launch type

Use these instructions if you are setting up App Mesh on a cluster running Amazon EKS with the Fargate launch type.

Configure IAM permissions

To set up IAM permissions, enter the following command. Replace *MyCluster* with the name of your cluster.

```
eksctl create iamserviceaccount --cluster MyCluster \
 --namespace howto-k8s-fargate \
 --name appmesh-pod \
 --attach-policy-arn arn:aws:iam::aws:policy/AWSAppMeshEnvoyAccess \
 --attach-policy-arn arn:aws:iam::aws:policy/AWSCloudMapDiscoverInstanceAccess \
 --attach-policy-arn arn:aws:iam::aws:policy/AWSXRayDaemonWriteAccess \
 --attach-policy-arn arn:aws:iam::aws:policy/CloudWatchLogsFullAccess \
 --attach-policy-arn arn:aws:iam::aws:policy/AWSAppMeshFullAccess \
 --attach-policy-arn arn:aws:iam::aws:policy/AWSCloudMapFullAccess \
 --override-existing-serviceaccounts \
 --approve
```

Install App Mesh

To install the App Mesh Kubernetes controller, follow the instructions in App Mesh Controller. Be sure to follow the instructions for Amazon EKS with the Fargate launch type.

Install a sample application

aws-app-mesh-examples contains several Kubernetes App Mesh walkthroughs. For this tutorial, you install a sample color application that works for Amazon EKS clusters with the Fargate launch type.

To use a sample App Mesh application to test Container Insights

1. Install the application using these instructions: https://github.com/aws/aws-app-mesh-examples/tree/main/walkthroughs/howto-k8s-fargate.

   Those instructions assume that you are creating a new cluster with the correct Fargate profile. If you want to use an Amazon EKS cluster that you've already set up, you can use the following commands to set up that cluster for this demonstration. Replace *MyCluster* with the name of your cluster.

   ```
   eksctl create iamserviceaccount --cluster MyCluster \
    --namespace howto-k8s-fargate \
    --name appmesh-pod \
    --attach-policy-arn arn:aws:iam::aws:policy/AWSAppMeshEnvoyAccess \
    --attach-policy-arn arn:aws:iam::aws:policy/AWSCloudMapDiscoverInstanceAccess \
    --attach-policy-arn arn:aws:iam::aws:policy/AWSXRayDaemonWriteAccess \
    --attach-policy-arn arn:aws:iam::aws:policy/CloudWatchLogsFullAccess \
    --attach-policy-arn arn:aws:iam::aws:policy/AWSAppMeshFullAccess \
    --attach-policy-arn arn:aws:iam::aws:policy/AWSCloudMapFullAccess \
   ```

```
 --override-existing-serviceaccounts \
 --approve
```

```
eksctl create fargateprofile --cluster MyCluster \
--namespace howto-k8s-fargate --name howto-k8s-fargate
```

2. Port forward the front application deployment:

```
kubectl -n howto-k8s-fargate port-forward deployment/front 8080:8080
```

3. Curl the front app:

```
while true; do  curl -s http://localhost:8080/color; sleep 0.1; echo ; done
```

4. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
5. In the Amazon Region where your cluster is running, choose **Metrics** in the navigation pane. The metric are in the **ContainerInsights/Prometheus** namespace.
6. To see the CloudWatch Logs events, choose **Log groups** in the navigation pane. The events are in the log group `/aws/containerinsights/your_cluster_name/prometheus` in the log stream `kubernetes-pod-appmesh-envoy`.

Deleting the App Mesh test environment

When you have finished using App Mesh and the sample application, use the following commands to delete the unnecessary resources. Delete the sample application by entering the following command:

```
cd aws-app-mesh-examples/walkthroughs/howto-k8s-fargate/
kubectl delete -f _output/manifest.yaml
```

Delete the App Mesh controller by entering the following command:

```
helm delete appmesh-controller -n appmesh-system
```

Set up NGINX with sample traffic on Amazon EKS and Kubernetes

NGINX is a web server that can also be used as a load balancer and reverse proxy. For more information about how Kubernetes uses NGINX for ingress , see kubernetes/ingress-nginx.

To install Ingress-NGINX with a sample traffic service to test Container Insights Prometheus support

1. Enter the following command to add the Helm ingress-nginx repo:

```
helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx
```

2. Enter the following commands:

```
kubectl create namespace nginx-ingress-sample

helm install my-nginx ingress-nginx/ingress-nginx \
--namespace nginx-ingress-sample \
--set controller.metrics.enabled=true \
--set-string controller.metrics.service.annotations."prometheus\.io/port"="10254" \
--set-string controller.metrics.service.annotations."prometheus\.io/scrape"="true"
```

3. Check whether the services started correctly by entering the following command:

```
kubectl get service -n nginx-ingress-sample
```

   The output of this command should display several columns, including an `EXTERNAL-IP` column.

4. Set an `EXTERNAL-IP` variable to the value of the `EXTERNAL-IP` column in the row of the NGINX ingress controller.

```
EXTERNAL_IP=your-nginx-controller-external-ip
```

5. Start some sample NGINX traffic by entering the following command.

```
SAMPLE_TRAFFIC_NAMESPACE=nginx-sample-traffic
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-
prometheus/sample_traffic/nginx-traffic/nginx-traffic-sample.yaml |
sed "s/{{external_ip}}/$EXTERNAL_IP/g" |
sed "s/{{namespace}}/$SAMPLE_TRAFFIC_NAMESPACE/g" |
kubectl apply -f -
```

6. Enter the following command to confirm that all three pods are in the `Running` status.

```
kubectl get pod -n $SAMPLE_TRAFFIC_NAMESPACE
```

If they are running, you should soon see metrics in the **ContainerInsights/Prometheus** namespace.

To uninstall NGINX and the sample traffic application

1. Delete the sample traffic service by entering the following command:

```
kubectl delete namespace $SAMPLE_TRAFFIC_NAMESPACE
```

2. Delete the NGINX egress by the Helm release name.

```
helm uninstall my-nginx --namespace nginx-ingress-sample
kubectl delete namespace nginx-ingress-sample
```

Set up memcached with a metric exporter on Amazon EKS and Kubernetes

memcached is an open-source memory object caching system. For more information, see [What is Memcached?](https://aws.amazon.com/memcached/).

If you are running memcached on a cluster with the Fargate launch type, you need to set up a Fargate profile before doing the steps in this procedure. To set up the profile, enter the following command. Replace *MyCluster* with the name of your cluster.

```
eksctl create fargateprofile --cluster MyCluster \
--namespace memcached-sample --name memcached-sample
```

To install memcached with a metric exporter to test Container Insights Prometheus support

1. Enter the following command to add the repo:

```
helm repo add bitnami https://charts.bitnami.com/bitnami
```

2. Enter the following command to create a new namespace:

```
kubectl create namespace memcached-sample
```

3. Enter the following command to install Memcached

```
helm install my-memcached bitnami/memcached --namespace memcached-sample \
--set metrics.enabled=true \
--set-string serviceAnnotations.prometheus\\.io/port="9150" \
--set-string serviceAnnotations.prometheus\\.io/scrape="true"
```

4. Enter the following command to confirm the annotation of the running service:

```
kubectl describe service my-memcached-metrics -n memcached-sample
```

You should see the following two annotations:

```
Annotations:   prometheus.io/port: 9150
               prometheus.io/scrape: true
```

To uninstall memcached

- Enter the following commands:

```
helm uninstall my-memcached --namespace memcached-sample
kubectl delete namespace memcached-sample
```

Set up Java/JMX sample workload on Amazon EKS and Kubernetes

JMX Exporter is an official Prometheus exporter that can scrape and expose JMX mBeans as Prometheus metrics. For more information, see prometheus/jmx_exporter.

Container Insights can collect predefined Prometheus metrics from Java Virtual Machine (JVM), Java, and Tomcat (Catalina) using the JMX Exporter.

Default Prometheus scrape configuration

By default, the CloudWatch agent with Prometheus support scrapes the Java/JMX Prometheus metrics from `http://CLUSTER_IP:9404/metrics` on each pod in an Amazon EKS or Kubernetes cluster. This is done by `role: pod` discovery of Prometheus `kubernetes_sd_config`. 9404 is the default port allocated for JMX Exporter by Prometheus. For more information about `role: pod` discovery, see [pod](). You can configure the JMX Exporter to expose the metrics on a different port or metrics_path. If you do change the port or path, update the default jmx scrape_config in the CloudWatch agent config map. Run the following command to get the current CloudWatch agent Prometheus configuration:

```
kubectl describe cm prometheus-config -n amazon-cloudwatch
```

The fields to change are the `/metrics` and `regex: '.*:9404$'` fields, as highlighted in the following example.

```
job_name: 'kubernetes-jmx-pod'
sample_limit: 10000
metrics_path: /metrics
kubernetes_sd_configs:
- role: pod
relabel_configs:
- source_labels: [__address__]
  action: keep
  regex: '.*:9404$'
- action: replace
  regex: (.+)
  source_labels:
```

Other Prometheus scrape configuration

If you expose your application running on a set of pods with Java/JMX Prometheus exporters by a Kubernetes Service, you can also switch to use `role: service` discovery or `role: endpoint` discovery of Prometheus `kubernetes_sd_config`. For more information about these discovery methods, see [service](), [endpoints](), and [<kubernetes_sd_config>.]().

More meta labels are provided by these two service discovery modes which could be useful for you to build the CloudWatch metrics dimensions. For example, you can relabel `__meta_kubernetes_service_name` to `Service` and include it into your metrics' dimension. For more informatio about customizing your CloudWatch metrics and their dimensions, see [CloudWatch agent configuration for Prometheus]().

Docker image with JMX Exporter

Next, build a Docker image. The following sections provide two example Dockerfiles.

When you have built the image, load it into Amazon EKS or Kubernetes, and then run the following command to verify that Prometheus metrics are exposed by JMX_EXPORTER on port 9404. Replace *$JAR_SAMPLE_TRAFFIC_POD* with the running pod name and replace *$JAR_SAMPLE_TRAFFIC_NAMESPACE* with your application namespace.

If you are running JMX Exporter on a cluster with the Fargate launch type, you also need to set up a Fargate profile before doing the steps in this procedure. To set up the profile, enter the following command. Replace *MyCluster* with the name of your cluster.

```
eksctl create fargateprofile --cluster MyCluster \
--namespace $JAR_SAMPLE_TRAFFIC_NAMESPACE\
 --name $JAR_SAMPLE_TRAFFIC_NAMESPACE
```

```
kubectl exec $JAR_SAMPLE_TRAFFIC_POD -n $JARCAT_SAMPLE_TRAFFIC_NAMESPACE -- curl
 http://localhost:9404
```

Example: Apache Tomcat Docker image with Prometheus metrics

Apache Tomcat server exposes JMX mBeans by default. You can integrate JMX Exporter with Tomcat to expose JMX mBeans as Prometheus metrics. The following example Dockerfile shows the steps to build a testing image:

```
# From Tomcat 9.0 JDK8 OpenJDK
FROM tomcat:9.0-jdk8-openjdk

RUN mkdir -p /opt/jmx_exporter

COPY ./jmx_prometheus_javaagent-0.12.0.jar /opt/jmx_exporter
COPY ./config.yaml /opt/jmx_exporter
COPY ./setenv.sh /usr/local/tomcat/bin
COPY your web application.war /usr/local/tomcat/webapps/

RUN chmod  o+x /usr/local/tomcat/bin/setenv.sh

ENTRYPOINT ["catalina.sh", "run"]
```

The following list explains the four COPY lines in this Dockerfile.

- Download the latest JMX Exporter jar file from https://github.com/prometheus/jmx_exporter.

- `config.yaml` is the JMX Exporter configuration file. For more information, see https://github.com/prometheus/jmx_exporter#Configuration.

  Here is a sample configuration file for Java and Tomcat:

```
lowercaseOutputName: true
lowercaseOutputLabelNames: true

rules:
- pattern: 'java.lang<type=OperatingSystem><>(FreePhysicalMemorySize|
TotalPhysicalMemorySize|FreeSwapSpaceSize|TotalSwapSpaceSize|SystemCpuLoad|
ProcessCpuLoad|OpenFileDescriptorCount|AvailableProcessors)'
  name: java_lang_OperatingSystem_$1
  type: GAUGE

- pattern: 'java.lang<type=Threading><>(TotalStartedThreadCount|ThreadCount)'
  name: java_lang_threading_$1
  type: GAUGE

- pattern: 'Catalina<type=GlobalRequestProcessor, name=\"(\w+-\w+)-(\d+)\"><>(\w+)'
  name: catalina_globalrequestprocessor_$3_total
  labels:
    port: "$2"
    protocol: "$1"
  help: Catalina global $3
  type: COUNTER

- pattern: 'Catalina<j2eeType=Servlet, WebModule=//([-a-zA-Z0-9+&@#/%?=~_|!:.,;]*[-
a-zA-Z0-9+&@#/%=~_|]), name=([-a-zA-Z0-9+/$%~_-|!.]*), J2EEApplication=none,
 J2EEServer=none><>(requestCount|maxTime|processingTime|errorCount)'
  name: catalina_servlet_$3_total
  labels:
    module: "$1"
    servlet: "$2"
  help: Catalina servlet $3 total
  type: COUNTER

- pattern: 'Catalina<type=ThreadPool, name="(\w+-\w+)-(\d+)"><>(currentThreadCount|
currentThreadsBusy|keepAliveCount|pollerThreadCount|connectionCount)'
  name: catalina_threadpool_$3
  labels:
    port: "$2"
    protocol: "$1"
```

```
  help: Catalina threadpool $3
  type: GAUGE

- pattern: 'Catalina<type=Manager, host=([-a-zA-Z0-9+&@#/%?=~_|!:.,;]*[-a-zA-
Z0-9+&@#/%=~_|]), context=([-a-zA-Z0-9+/$%~_-|!.]*)><>(processingTime|sessionCounter|
rejectedSessions|expiredSessions)'
  name: catalina_session_$3_total
  labels:
    context: "$2"
    host: "$1"
  help: Catalina session $3 total
  type: COUNTER

- pattern: ".*"
```

- `setenv.sh` is a Tomcat startup script to start the JMX exporter along with Tomcat and expose Prometheus metrics on port 9404 of the localhost. It also provides the JMX Exporter with the `config.yaml` file path.

```
$ cat setenv.sh
export JAVA_OPTS="-javaagent:/opt/jmx_exporter/
jmx_prometheus_javaagent-0.12.0.jar=9404:/opt/jmx_exporter/config.yaml $JAVA_OPTS"
```

- your web application.war is your web application `war` file to be loaded by Tomcat.

Build a Docker image with this configuration and upload it to an image repository.

Example: Java Jar Application Docker image with Prometheus metrics

The following example Dockerfile shows the steps to build a testing image:

```
# Alpine Linux with OpenJDK JRE
FROM openjdk:8-jre-alpine

RUN mkdir -p /opt/jmx_exporter

COPY ./jmx_prometheus_javaagent-0.12.0.jar /opt/jmx_exporter
COPY ./SampleJavaApplication-1.0-SNAPSHOT.jar /opt/jmx_exporter
COPY ./start_exporter_example.sh /opt/jmx_exporter
COPY ./config.yaml /opt/jmx_exporter

RUN chmod -R o+x /opt/jmx_exporter
```

```
RUN apk add curl

ENTRYPOINT exec /opt/jmx_exporter/start_exporter_example.sh
```

The following list explains the four `COPY` lines in this Dockerfile.

- Download the latest JMX Exporter jar file from https://github.com/prometheus/jmx_exporter.
- `config.yaml` is the JMX Exporter configuration file. For more information, see https://github.com/prometheus/jmx_exporter#Configuration.

  Here is a sample configuration file for Java and Tomcat:

```
lowercaseOutputName: true
lowercaseOutputLabelNames: true

rules:
- pattern: 'java.lang<type=OperatingSystem><>(FreePhysicalMemorySize|
TotalPhysicalMemorySize|FreeSwapSpaceSize|TotalSwapSpaceSize|SystemCpuLoad|
ProcessCpuLoad|OpenFileDescriptorCount|AvailableProcessors)'
  name: java_lang_OperatingSystem_$1
  type: GAUGE

- pattern: 'java.lang<type=Threading><>(TotalStartedThreadCount|ThreadCount)'
  name: java_lang_threading_$1
  type: GAUGE

- pattern: 'Catalina<type=GlobalRequestProcessor, name=\"(\w+-\w+)-(\d+)\"><>(\w+)'
  name: catalina_globalrequestprocessor_$3_total
  labels:
    port: "$2"
    protocol: "$1"
  help: Catalina global $3
  type: COUNTER

- pattern: 'Catalina<j2eeType=Servlet, WebModule=//([-a-zA-Z0-9+&@#/%?=~_|!:.,;]*[-
a-zA-Z0-9+&@#/%=~_|]), name=([-a-zA-Z0-9+/$%~_-|!.]*), J2EEApplication=none,
 J2EEServer=none><>(requestCount|maxTime|processingTime|errorCount)'
  name: catalina_servlet_$3_total
  labels:
    module: "$1"
    servlet: "$2"
  help: Catalina servlet $3 total
  type: COUNTER
```

```
- pattern: 'Catalina<type=ThreadPool, name="(\w+-\w+)-(\d+)"><>(currentThreadCount|
currentThreadsBusy|keepAliveCount|pollerThreadCount|connectionCount)'
  name: catalina_threadpool_$3
  labels:
    port: "$2"
    protocol: "$1"
  help: Catalina threadpool $3
  type: GAUGE

- pattern: 'Catalina<type=Manager, host=([-a-zA-Z0-9+&@#/%?=~_|!:.,;]*[-a-zA-
Z0-9+&@#/%=~_|]), context=([-a-zA-Z0-9+/$%~_-|!.]*)><>(processingTime|sessionCounter|
rejectedSessions|expiredSessions)'
  name: catalina_session_$3_total
  labels:
    context: "$2"
    host: "$1"
  help: Catalina session $3 total
  type: COUNTER

- pattern: ".*"
```

- `start_exporter_example.sh` is the script to start the JAR application with the Prometheus metrics exported. It also provides the JMX Exporter with the `config.yaml` file path.

```
$ cat start_exporter_example.sh
java -javaagent:/opt/jmx_exporter/jmx_prometheus_javaagent-0.12.0.jar=9404:/
opt/jmx_exporter/config.yaml -cp  /opt/jmx_exporter/SampleJavaApplication-1.0-
SNAPSHOT.jar com.gubupt.sample.app.App
```

- SampleJavaApplication-1.0-SNAPSHOT.jar is the sample Java application jar file. Replace it with the Java application that you want to monitor.

Build a Docker image with this configuration and upload it to an image repository.

Set up HAProxy with a metric exporter on Amazon EKS and Kubernetes

HAProxy is an open-source proxy application. For more information, see [HAProxy](https://www.haproxy.org/).

If you are running HAProxy on a cluster with the Fargate launch type, you need to set up a Fargate profile before doing the steps in this procedure. To set up the profile, enter the following command. Replace *`MyCluster`* with the name of your cluster.

```
eksctl create fargateprofile --cluster MyCluster \
--namespace haproxy-ingress-sample --name haproxy-ingress-sample
```

To install HAProxy with a metric exporter to test Container Insights Prometheus support

1. Enter the following command to add the Helm incubator repo:

```
helm repo add haproxy-ingress https://haproxy-ingress.github.io/charts
```

2. Enter the following command to create a new namespace:

```
kubectl create namespace haproxy-ingress-sample
```

3. Enter the following commands to install HAProxy:

```
helm install haproxy haproxy-ingress/haproxy-ingress \
--namespace haproxy-ingress-sample \
--set defaultBackend.enabled=true \
--set controller.stats.enabled=true \
--set controller.metrics.enabled=true \
--set-string controller.metrics.service.annotations."prometheus\.io/port"="9101" \
--set-string controller.metrics.service.annotations."prometheus\.io/scrape"="true"
```

4. Enter the following command to confirm the annotation of the service:

```
kubectl describe service haproxy-haproxy-ingress-metrics -n haproxy-ingress-sample
```

You should see the following annotations.

```
Annotations:   prometheus.io/port: 9101
               prometheus.io/scrape: true
```

To uninstall HAProxy

- Enter the following commands:

```
helm uninstall haproxy --namespace haproxy-ingress-sample
kubectl delete namespace haproxy-ingress-sample
```

Tutorial for adding a new Prometheus scrape target: Redis OSS on Amazon EKS and Kubernetes clusters

This tutorial provides a hands-on introduction to scrape the Prometheus metrics of a sample Redis OSS application on Amazon EKS and Kubernetes. Redis OSS (https://redis.io/) is an open source (BSD licensed), in-memory data structure store, used as a database, cache and message broker. For more information, see redis.

redis_exporter (MIT License licensed) is used to expose the Redis OSS Prometheus metrics on the specified port (default: 0.0.0.0:9121). For more information, see redis_exporter.

The Docker images in the following two Docker Hub repositories are used in this tutorial:

- redis
- redis_exporter

To install a sample Redis OSS workload which exposes Prometheus metrics

1. Set the namespace for the sample Redis OSS workload.

   ```
   REDIS_NAMESPACE=redis-sample
   ```

2. If you are running Redis OSS on a cluster with the Fargate launch type, you need to set up a Fargate profile. To set up the profile, enter the following command. Replace *MyCluster* with the name of your cluster.

   ```
   eksctl create fargateprofile --cluster MyCluster \
   --namespace $REDIS_NAMESPACE --name $REDIS_NAMESPACE
   ```

3. Enter the following command to install the sample Redis OSS workload.

   ```
   curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
   insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-
   prometheus/sample_traffic/redis/redis-traffic-sample.yaml \
   | sed "s/{{namespace}}/$REDIS_NAMESPACE/g" \
   | kubectl apply -f -
   ```

4. The installation includes a service named `my-redis-metrics` which exposes the Redis OSS Prometheus metric on port 9121 Enter the following command to get the details of the service:

```
kubectl describe service/my-redis-metrics  -n $REDIS_NAMESPACE
```

In the `Annotations` section of the results, you'll see two annotations which match the Prometheus scrape configuration of the CloudWatch agent, so that it can auto-discover the workloads:

```
prometheus.io/port: 9121
prometheus.io/scrape: true
```

The related Prometheus scrape configuration can be found in the `- job_name: kubernetes-service-endpoints` section of `kubernetes-eks.yaml` or `kubernetes-k8s.yaml`.

To start collecting Redis OSS Prometheus metrics in CloudWatch

1. Download the latest version of the of `kubernetes-eks.yaml` or `kubernetes-k8s.yaml` file by entering one of the following commands. For an Amazon EKS cluster with the EC2 launch type, enter this command.

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-prometheus/prometheus-eks.yaml
```

For an Amazon EKS cluster with the Fargate launch type, enter this command.

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-prometheus/prometheus-eks-fargate.yaml
```

For a Kubernetes cluster running on an Amazon EC2 instance, enter this command.

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-prometheus/prometheus-k8s.yaml
```

2. Open the file with a text editor, and find the `cwagentconfig.json` section. Add the following subsection and save the changes. Be sure that the indentation follows the existing pattern.

```
{
  "source_labels": ["pod_name"],
  "label_matcher": "^redis-instance$",
  "dimensions": [["Namespace","ClusterName"]],
  "metric_selectors": [
    "^redis_net_(in|out)put_bytes_total$",
    "^redis_(expired|evicted)_keys_total$",
    "^redis_keyspace_(hits|misses)_total$",
    "^redis_memory_used_bytes$",
    "^redis_connected_clients$"
  ]
},
{
  "source_labels": ["pod_name"],
  "label_matcher": "^redis-instance$",
  "dimensions": [["Namespace","ClusterName","cmd"]],
  "metric_selectors": [
    "^redis_commands_total$"
  ]
},
{
  "source_labels": ["pod_name"],
  "label_matcher": "^redis-instance$",
  "dimensions": [["Namespace","ClusterName","db"]],
  "metric_selectors": [
    "^redis_db_keys$"
  ]
},
```

   The section you added puts the Redis OSS metrics onto the CloudWatch agent allow list. For a list of these metrics, see the following section.

3. If you already have the CloudWatch agent with Prometheus support deployed in this cluster, you must delete it by entering the following command.

```
kubectl delete deployment cwagent-prometheus -n amazon-cloudwatch
```

4. Deploy the CloudWatch agent with your updated configuration by entering one of the following commands. Replace *MyCluster* and *region* to match your settings.

   For an Amazon EKS cluster with the EC2 launch type, enter this command.

```
kubectl apply -f prometheus-eks.yaml
```

   For an Amazon EKS cluster with the Fargate launch type, enter this command.

```
cat prometheus-eks-fargate.yaml \
| sed "s/{{cluster_name}}/MyCluster/;s/{{region_name}}/region/" \
| kubectl apply -f -
```

   For a Kubernetes cluster, enter this command.

```
cat prometheus-k8s.yaml \
| sed "s/{{cluster_name}}/MyCluster/;s/{{region_name}}/region/" \
| kubectl apply -f -
```

Viewing your Redis OSS Prometheus metrics

This tutorial sends the following metrics to the **ContainerInsights/Prometheus** namespace in CloudWatch. You can use the CloudWatch console to see the metrics in that namespace.

Metric name	Dimensions	
`redis_net_input_bytes_total`	ClusterName, `Namespace`	
`redis_net_output_bytes_total`	ClusterName, `Namespace`	
`redis_expired_keys_total`	ClusterName, `Namespace`	

Metric name	Dimensions	
redis_evicted_keys_total	ClusterName, Namespace	
redis_keyspace_hits_total	ClusterName, Namespace	
redis_keyspace_misses_total	ClusterName, Namespace	
redis_memory_used_bytes	ClusterName, Namespace	
redis_connected_clients	ClusterName, Namespace	
redis_commands_total	ClusterName, Namespace , cmd	
redis_db_keys	ClusterName, Namespace , db	

Note

The value of the **cmd** dimension can be: append, client, command, config, dbsize, flushall, get, incr, info, latency, or slowlog.

The value of the **db** dimension can be db0 to db15.

You can also create a CloudWatch dashboard for your Redis OSS Prometheus metrics.

To create a dashboard for Redis OSS Prometheus metrics

1. Create environment variables, replacing the values below to match your deployment.

```
DASHBOARD_NAME=your_cw_dashboard_name
REGION_NAME=your_metric_region_such_as_us-east-1
CLUSTER_NAME=your_k8s_cluster_name_here
NAMESPACE=your_redis_service_namespace_here
```

2. Enter the following command to create the dashboard.

```
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-
prometheus/sample_cloudwatch_dashboards/redis/cw_dashboard_redis.json \
| sed "s/{{YOUR_AWS_REGION}}/${REGION_NAME}/g" \
| sed "s/{{YOUR_CLUSTER_NAME}}/${CLUSTER_NAME}/g" \
| sed "s/{{YOUR_NAMESPACE}}/${NAMESPACE}/g" \
```

Prometheus metric type conversion by the CloudWatch Agent

The Prometheus client libraries offer four core metric types:

- Counter
- Gauge
- Summary
- Histogram

The CloudWatch agent supports the counter, gauge, and summary metric types.

The Prometheus metrics with the unsupported histogram metric type are dropped by the CloudWatch agent. For more information, see Logging dropped Prometheus metrics.

Gauge metrics

A Prometheus gauge metric is a metric that represents a single numerical value that can arbitrarily go up and down. The CloudWatch agent scrapes gauge metrics and send these values out directly.

Counter metrics

A Prometheus counter metric is a cumulative metric that represents a single monotonically increasing counter whose value can only increase or be reset to zero. The CloudWatch agent calculates a delta from the previous scrape and sends the delta value as the metric value in the log event. So the CloudWatch agent will start to produce one log event from the second scrape and continue with subsequent scrapes, if any.

Summary metrics

A Prometheus summary metric is a complex metric type which is represented by multiple data points. It provides a total count of observations and a sum of all observed values. It calculates configurable quantiles over a sliding time window.

The sum and count of a summary metric are cumulative, but the quantiles are not. The following example shows the variance of quantiles.

```
# TYPE go_gc_duration_seconds summary
go_gc_duration_seconds{quantile="0"} 7.123e-06
go_gc_duration_seconds{quantile="0.25"} 9.204e-06
go_gc_duration_seconds{quantile="0.5"} 1.1065e-05
go_gc_duration_seconds{quantile="0.75"} 2.8731e-05
go_gc_duration_seconds{quantile="1"} 0.003841496
go_gc_duration_seconds_sum 0.37630427
go_gc_duration_seconds_count 9774
```

The CloudWatch agent handles the sum and count of a summary metric in the same way as it handles counter metrics, as described in the previous section. The CloudWatch agent preserves the quantile values as they are originally reported.

Prometheus metrics collected by the CloudWatch agent

The CloudWatch agent with Prometheus support automatically collects metrics from several services and workloads. The metrics that are collected by default are listed in the following sections. You can also configure the agent to collect more metrics from these services, and to collect Prometheus metrics from other applications and services. For more information about collecting additional metrics, see CloudWatch agent configuration for Prometheus.

Prometheus metrics collected from Amazon EKS and Kubernetes clusters are in the **ContainerInsights/Prometheus** namespace. Prometheus metrics collected from Amazon ECS clusters are in the **ECS/ContainerInsights/Prometheus** namespace.

Topics

- Prometheus metrics for App Mesh

- Prometheus metrics for NGINX
- Prometheus metrics for Memcached
- Prometheus metrics for Java/JMX
- Prometheus metrics for HAProxy

Prometheus metrics for App Mesh

The following metrics are automatically collected from App Mesh .

Prometheus metrics for App Mesh on Amazon EKS and Kubernetes clusters

Metric name	Dimensions	
envoy_http_downstream_rq_total	ClusterName, Namespace	
envoy_http_downstream_rq_xx	ClusterName, Namespace ClusterName, Namespace , envoy_http_conn_manager_prefix, envoy_response_code_class	
envoy_cluster_upstream_cx_rx_bytes_total	ClusterName, Namespace	
envoy_cluster_upstream_cx_tx_bytes_total	ClusterName, Namespace	
envoy_cluster_membership_healthy	ClusterName, Namespace	

Metric name	Dimensions	
envoy_cluster_membership_total	ClusterName, Namespace	
envoy_server_memory_heap_size	ClusterName, Namespace	
envoy_server_memory_allocated	ClusterName, Namespace	
envoy_cluster_upstream_cx_connect_timeout	ClusterName, Namespace	
envoy_cluster_upstream_rq_pending_failure_eject	ClusterName, Namespace	
envoy_cluster_upstream_rq_pending_overflow	ClusterName, Namespace	
envoy_cluster_upstream_rq_timeout	ClusterName, Namespace	

Metric name	Dimensions	
envoy_cluster_upstream_rq_try_per_timeout	ClusterName, Namespace	
envoy_cluster_upstream_rq_rx_reset	ClusterName, Namespace	
envoy_cluster_upstream_cx_destroy_local_with_active_rq	ClusterName, Namespace	
envoy_cluster_upstream_cx_destroy_remote_active_rq	ClusterName, Namespace	
envoy_cluster_upstream_rq_maintenance_mode	ClusterName, Namespace	
envoy_cluster_upstream_flow_control_paused_reading_total	ClusterName, Namespace	

Metric name	Dimensions	
envoy_cluster_upstream_flow_control_resumed_reading_total	ClusterName, Namespace	
envoy_cluster_upstream_flow_control_backed_up_total	ClusterName, Namespace	
envoy_cluster_upstream_flow_control_drained_total	ClusterName, Namespace	
envoy_cluster_upstream_rq_retry	ClusterName, Namespace	
envoy_cluster_upstream_rq_retry_success	ClusterName, Namespace	
envoy_cluster_upstream_rq_retry_overflow	ClusterName, Namespace	
envoy_server_live	ClusterName, Namespace	

Metric name	Dimensions	
envoy_server_uptime	ClusterName, Namespace	

Prometheus metrics for App Mesh on Amazon ECS clusters

Metric name	Dimensions	
envoy_http_downstream_rq_total	ClusterName, TaskDefinitionFamily	
envoy_http_downstream_rq_xx	ClusterName, TaskDefinitionFamily	
envoy_cluster_upstream_cx_rx_bytes_total	ClusterName, TaskDefinitionFamily	
envoy_cluster_upstream_cx_tx_bytes_total	ClusterName, TaskDefinitionFamily	
envoy_cluster_membership_healthy	ClusterName, TaskDefinitionFamily	
envoy_cluster_membership_total	ClusterName, TaskDefinitionFamily	

Metric name	Dimensions	
envoy_server_memory_heap_size	ClusterName, TaskDefinitionFamily	
envoy_server_memory_allocated	ClusterName, TaskDefinitionFamily	
envoy_cluster_upstream_cx_connect_timeout	ClusterName, TaskDefinitionFamily	
envoy_cluster_upstream_rq_pending_failure_eject	ClusterName, TaskDefinitionFamily	
envoy_cluster_upstream_rq_pending_overflow	ClusterName, TaskDefinitionFamily	
envoy_cluster_upstream_rq_timeout	ClusterName, TaskDefinitionFamily	
envoy_cluster_upstream_rq_try_per_timeout	ClusterName, TaskDefinitionFamily	

Metric name	Dimensions	
envoy_cluster_upstream_rq_rx_reset	ClusterName, TaskDefinitionFamily	
envoy_cluster_upstream_cx_destroy_local_with_active_rq	ClusterName, TaskDefinitionFamily	
envoy_cluster_upstream_cx_destroy_remote_active_rq	ClusterName, TaskDefinitionFamily	
envoy_cluster_upstream_rq_maintenance_mode	ClusterName, TaskDefinitionFamily	
envoy_cluster_upstream_flow_control_paused_reading_total	ClusterName, TaskDefinitionFamily	

Metric name	Dimensions	
envoy_cluster_upstream_flow_control_resumed_reading_total	ClusterName, TaskDefinitionFamily	
envoy_cluster_upstream_flow_control_backed_up_total	ClusterName, TaskDefinitionFamily	
envoy_cluster_upstream_flow_control_drained_total	ClusterName, TaskDefinitionFamily	
envoy_cluster_upstream_rq_retry	ClusterName, TaskDefinitionFamily	
envoy_cluster_upstream_rq_retry_success	ClusterName, TaskDefinitionFamily	
envoy_cluster_upstream_rq_retry_overflow	ClusterName, TaskDefinitionFamily	
envoy_server_live	ClusterName, TaskDefinitionFamily	

Metric name	Dimensions	
envoy_server_uptime	ClusterName, TaskDefinitionFamily	
envoy_http_downstream_rq_xx	ClusterName, TaskDefinitionFamily, envoy_http_conn_manager_prefix, envoy_response_code_class ClusterName, TaskDefinitionFamily, envoy_response_code_class	

Note

TaskDefinitionFamily is the Kubernetes namespace of the mesh.
The value of envoy_http_conn_manager_prefix can be ingress, egress, or admin.
The value of envoy_response_code_class can be 1 (stands for 1xx), 2 stands for 2xx), 3 stands for 3xx), 4 stands for 4xx), or 5 stands for 5xx).

Prometheus metrics for NGINX

The following metrics are automatically collected from NGINX on Amazon EKS and Kubernetes clusters.

Metric name	Dimensions	
nginx_ingress_controller_nginx_process_cpu_seconds_total	ClusterName, Namespace , Service	
nginx_ingress_controller_success	ClusterName, Namespace , Service	

Metric name	Dimensions	
nginx_ingress_controller_requests	ClusterName, Namespace , Service	
nginx_ingress_controller_nginx_process_connections	ClusterName, Namespace , Service	
nginx_ingress_controller_nginx_process_connections_total	ClusterName, Namespace , Service	
nginx_ingress_controller_nginx_process_resident_memory_bytes	ClusterName, Namespace , Service	
nginx_ingress_controller_config_last_reload_successful	ClusterName, Namespace , Service	

Metric name	Dimensions	
nginx_ingress_controller_requests	ClusterName, Namespace , Service, status	

Prometheus metrics for Memcached

The following metrics are automatically collected from Memcached on Amazon EKS and Kubernetes clusters.

Metric name	Dimensions	
memcached_current_items	ClusterName, Namespace , Service	
memcached_current_connections	ClusterName, Namespace , Service	
memcached_limit_bytes	ClusterName, Namespace , Service	
memcached_current_bytes	ClusterName, Namespace , Service	
memcached_written_bytes_total	ClusterName, Namespace , Service	
memcached_read_bytes_total	ClusterName, Namespace , Service	
memcached_items_evicted_total	ClusterName, Namespace , Service	

Metric name	Dimensions
memcached_items_reclaimed_total	ClusterName, Namespace , Service
memcached_commands_total	ClusterName, Namespace , Service ClusterName, Namespace , Service, command ClusterName, Namespace , Service, status, command

Prometheus metrics for Java/JMX

Metrics collected on Amazon EKS and Kubernetes clusters

On Amazon EKS and Kubernetes clusters, Container Insights can collect the following predefined Prometheus metrics from the Java Virtual Machine (JVM), Java, and Tomcat (Catalina) using the JMX Exporter. For more information, see [prometheus/jmx_exporter](https://github.com/prometheus/jmx_exporter) on Github.

Java/JMX on Amazon EKS and Kubernetes clusters

Metric name	Dimensions
jvm_classes_loaded	ClusterName , Namespace
jvm_threads_current	ClusterName , Namespace
jvm_threads_daemon	ClusterName , Namespace
java_lang_operatingsystem_totalswapspacesize	ClusterName , Namespace

Metric name	Dimensions	
java_lang_operatingsystem_systemcpuload	ClusterName ,Namespace	
java_lang_operatingsystem_processcpuload	ClusterName ,Namespace	
java_lang_operatingsystem_freeswapspacesize	ClusterName ,Namespace	
java_lang_operatingsystem_totalphysicalmemorysize	ClusterName ,Namespace	
java_lang_operatingsystem_freephysicalmemorysize	ClusterName ,Namespace	
java_lang_operatingsystem_openfiledescriptorcount	ClusterName ,Namespace	

Metric name	Dimensions	
java_lang_operatingsystem_availableprocessors	ClusterName , Namespace	
jvm_memory_bytes_used	ClusterName , Namespace , area	
jvm_memory_pool_bytes_used	ClusterName , Namespace , pool	

Note

The values of the area dimension can be heap or nonheap.

The values of the pool dimension can be Tenured Gen, Compress Class Space, Survivor Space, Eden Space, Code Cache, or Metaspace.

Tomcat/JMX on Amazon EKS and Kubernetes clusters

In addition to the Java/JMX metrics in the previous table, the following metrics are also collected for the Tomcat workload.

Metric name	Dimensions	
catalina_manager_activesessions	ClusterName , Namespace	
catalina_manager_rejectedsessions	ClusterName , Namespace	

Metric name	Dimensions	
catalina_globalrequestprocessor_bytesreceived	ClusterName ,Namespace	
catalina_globalrequestprocessor_bytessent	ClusterName ,Namespace	
catalina_globalrequestprocessor_requestcount	ClusterName ,Namespace	
catalina_globalrequestprocessor_errorcount	ClusterName ,Namespace	
catalina_globalrequestprocessor_processingtime	ClusterName ,Namespace	

Java/JMX on Amazon ECS clusters

Metric name	Dimensions	
jvm_classes_loaded	ClusterName ,TaskDefinitionFamily	
jvm_threads_current	ClusterName ,TaskDefinitionFamily	
jvm_threads_daemon	ClusterName ,TaskDefinitionFamily	
java_lang_operatingsystem_totalswapspacesize	ClusterName ,TaskDefinitionFamily	
java_lang_operatingsystem_systemcpuload	ClusterName ,TaskDefinitionFamily	
java_lang_operatingsystem_processcpuload	ClusterName ,TaskDefinitionFamily	
java_lang_operatingsystem_freeswapspacesize	ClusterName ,TaskDefinitionFamily	
java_lang_operatingsystem_totalphysicalmemorysize	ClusterName ,TaskDefinitionFamily	

Metric name	Dimensions	
java_lang_operatingsystem_freephysicalmemorysize	ClusterName , TaskDefinitionFamily	
java_lang_operatingsystem_openfiledescriptorcount	ClusterName , TaskDefinitionFamily	
java_lang_operatingsystem_availableprocessors	ClusterName , TaskDefinitionFamily	
jvm_memory_bytes_used	ClusterName , TaskDefinitionFamily, area	
jvm_memory_pool_bytes_used	ClusterName , TaskDefinitionFamily, pool	

Note

The values of the area dimension can be heap or nonheap.

The values of the pool dimension can be Tenured Gen, Compress Class Space, Survivor Space, Eden Space, Code Cache, or Metaspace.

Tomcat/JMX on Amazon ECS clusters

In addition to the Java/JMX metrics in the previous table, the following metrics are also collected for the Tomcat workload on Amazon ECS clusters.

Metric name	Dimensions	
catalina_manager_activesessions	ClusterName , TaskDefinitionFamily	
catalina_manager_rejectedsessions	ClusterName , TaskDefinitionFamily	
catalina_globalrequestprocessor_bytesreceived	ClusterName , TaskDefinitionFamily	
catalina_globalrequestprocessor_bytessent	ClusterName , TaskDefinitionFamily	
catalina_globalrequestprocessor_requestcount	ClusterName , TaskDefinitionFamily	
catalina_globalrequestprocessor_errorcount	ClusterName , TaskDefinitionFamily	

Metric name	Dimensions	
catalina_globalrequestprocessor_processingtime	ClusterName ,TaskDefinitionFamily	

Prometheus metrics for HAProxy

The following metrics are automatically collected from HAProxy on Amazon EKS and Kubernetes clusters.

The metrics collected depend on which version of HAProxy Ingress that you are using. For more information about HAProxy Ingress and its versions, see haproxy-ingress.

Metric name	Dimensions	Availability
haproxy_backend_bytes_in_total	ClusterName ,Namespace , Service	All versions of HAProxy Ingress
haproxy_backend_bytes_out_total	ClusterName ,Namespace , Service	All versions of HAProxy Ingress
haproxy_backend_connection_errors_total	ClusterName ,Namespace , Service	All versions of HAProxy Ingress
haproxy_backend_connections_total	ClusterName ,Namespace , Service	All versions of HAProxy Ingress

Metric name	Dimensions	Availability
haproxy_backend_current_sessions	ClusterName , Namespace , Service	All versions of HAProxy Ingress
haproxy_backend_http_responses_total	ClusterName , Namespace , Service, code, backend	All versions of HAProxy Ingress
haproxy_backend_status	ClusterName , Namespace , Service	Only in versions 0.10 or later of HAProxy Ingress
haproxy_backend_up	ClusterName , Namespace , Service	Only in versions of HAProxy Ingress earlier than 0.10
haproxy_frontend_bytes_in_total	ClusterName , Namespace , Service	All versions of HAProxy Ingress
haproxy_frontend_bytes_out_total	ClusterName , Namespace , Service	All versions of HAProxy Ingress
haproxy_frontend_connections_total	ClusterName , Namespace , Service	All versions of HAProxy Ingress
haproxy_frontend_current_sessions	ClusterName , Namespace , Service	All versions of HAProxy Ingress

Metric name	Dimensions	Availability
haproxy_frontend_http_requests_total	ClusterName , Namespace , Service	All versions of HAProxy Ingress
haproxy_frontend_http_responses_total	ClusterName , Namespace , Service, code, frontend	All versions of HAProxy Ingress
haproxy_frontend_request_errors_total	ClusterName , Namespace , Service	All versions of HAProxy Ingress
haproxy_frontend_requests_denied_total	ClusterName , Namespace , Service	All versions of HAProxy Ingress

Note

The values of the code dimension can be 1xx, 2xx, 3xx, 4xx, 5xx, or other.
The values of the backend dimension can be:

- http-default-backend, http-shared-backend, or httpsback-shared-backend for HAProxy Ingress version 0.0.27 or earlier.
- _default_backend for HAProxy Ingress versions later than 0.0.27.

The values of the frontend dimension can be:

- httpfront-default-backend, httpfront-shared-frontend, or httpfronts for HAProxy Ingress version 0.0.27 or earlier.
- _front_http or _front_https for HAProxy Ingress versions later than 0.0.27.

Viewing your Prometheus metrics

You can monitor and alarm on all your Prometheus metrics including the curated pre-aggregated metrics from App Mesh, NGINX, Java/JMX, Memcached, and HAProxy, and any other manually configured Prometheus exporter you may have added. For more information about collecting metrics from other Prometheus exporters, see Tutorial for adding a new Prometheus scrape target: Prometheus API Server metrics.

In the CloudWatch console, Container Insights provides the following pre-built reports:

- For Amazon EKS and Kubernetes clusters, there are pre-built reports for App Mesh, NGINX, HAPROXY, Memcached, and Java/JMX.
- For Amazon ECS clusters, there are pre-built reports for App Mesh and Java/JMX.

Container Insights also provides custom dashboards for each of the workloads that Container Insights collects curated metrics from. You can download these dashboards from GitHub

To see all your Prometheus metrics

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Metrics**.
3. In the list of namespaces, choose **ContainerInsights/Prometheus** or **ECS/ContainerInsights/Prometheus**.
4. Choose one of the sets of dimensions in the following list. Then select the checkbox next to the metrics that you want to see.

To see pre-built reports on your Prometheus metrics

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Performance Monitoring**.
3. In the drop-down box near the top of the page, choose any of the Prometheus options.

   In the other drop-down box, choose a cluster to view

We have also provided custom dashboards for NGINX, App Mesh, Memcached, HAProxy, and Java/JMX.

To use a custom dashboard that Amazon has provided

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Dashboards**.
3. Choose **Create Dashboard**. Enter a name for the new dashboard, and choose **Create dashboard**.
4. In **Add to this dashboard**, choose **Cancel**.
5. Choose **Actions**, **View/edit source**.
6. Download one of the following JSON files:
   - NGINX custom dashboard source on Github.
   - App Mesh custom dashboard source on Github.
   - Memcached custom dashboard source on Github
   - HAProxy-Ingress custom dashboard source on Github
   - Java/JMX custom dashboard source on Github.
7. Open the JSON file that you downloaded with a text editor, and make the following changes:
   - Replace all the `{{YOUR_CLUSTER_NAME}}` strings with the exact name of your cluster. Make sure not to add whitespaces before or after the text.
   - Replace all the `{{YOUR_REGION}}` strings with the Amazon Region where your cluster is running. For example, **`us-west-1`** Make sure not to add whitespaces before or after the text.
   - Replace all the `{{YOUR_NAMESPACE}}` strings with the exact namespace of your workload.
   - Replace all the `{{YOUR_SERVICE_NAME}}` strings with the exact service name of your workload. For example, **`haproxy-haproxy-ingress-controller-metrics`**
8. Copy the entire JSON blob and paste it into the text box in the CloudWatch console, replacing what is already in the box.
9. Choose **Update**, **Save dashboard**.

Prometheus metrics troubleshooting

This section provides help for troubleshooting your Prometheus metrics setup.

Topics

- [Prometheus metrics troubleshooting on Amazon ECS]
- [Prometheus metrics troubleshooting on Amazon EKS and Kubernetes clusters]

Prometheus metrics troubleshooting on Amazon ECS

This section provides help for troubleshooting your Prometheus metrics setup on Amazon ECS clusters.

I don't see Prometheus metrics sent to CloudWatch Logs

The Prometheus metrics should be ingested as log events in the log group **/aws/ecs/containerinsights/cluster-name/Prometheus**. If the log group is not created or the Prometheus metrics are not sent to the log group, you will need to first check whether the Prometheus targets have been successfully discovered by the CloudWatch agent. And next check the security group and permission settings of the CloudWatch agent. The following steps guide you to do the debugging.

Step 1: Enable the CloudWatch agent debugging mode

First, change the CloudWatch agent to debug mode by adding the following bold lines to your Amazon CloudFormation template file, `cwagent-ecs-prometheus-metric-for-bridge-host.yaml` or `cwagent-ecs-prometheus-metric-for-awsvpc.yaml`. Then save the file.

```
cwagentconfig.json: |
    {
      "agent": {
        "debug": true
      },
      "logs": {
        "metrics_collected": {
```

Create a new Amazon CloudFormation changeset against the existing stack. Set other parameters in the changeset to the same values as in your existing Amazon CloudFormation stack. The following example is for a CloudWatch agent installed in an Amazon ECS cluster using the EC2 launch type and the bridge network mode.

```
ECS_NETWORK_MODE=bridge
 CREATE_IAM_ROLES=True
ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name
NEW_CHANGESET_NAME=your_selected_ecs_execution_role_name
```

```
aws cloudformation create-change-set --stack-name CWAgent-Prometheus-ECS-${ECS_CLUSTER_NAME}-EC2-${ECS_NETWORK_MODE} \
    --template-body file://cwagent-ecs-prometheus-metric-for-bridge-host.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=$ECS_CLUSTER_NAME \
                 ParameterKey=CreateIAMRoles,ParameterValue=$CREATE_IAM_ROLES \
                 ParameterKey=ECSNetworkMode,ParameterValue=$ECS_NETWORK_MODE \
                 ParameterKey=TaskRoleName,ParameterValue=$ECS_TASK_ROLE_NAME \
                 ParameterKey=ExecutionRoleName,ParameterValue=$ECS_EXECUTION_ROLE_NAME \
    --capabilities CAPABILITY_NAMED_IAM \
    --region $AWS_REGION \
    --change-set-name $NEW_CHANGESET_NAME
```

Go to the Amazon CloudFormation console to review the new changeset, `$NEW_CHANGESET_NAME`. There should be one change applied to the **CWAgentConfigSSMParameter** resource. Execute the changeset and restart the CloudWatch agent task by entering the following commands.

```
aws ecs update-service --cluster $ECS_CLUSTER_NAME \
--desired-count 0 \
--service your_service_name_here \
--region $AWS_REGION
```

Wait about 10 seconds and then enter the following command.

```
aws ecs update-service --cluster $ECS_CLUSTER_NAME \
--desired-count 1 \
--service your_service_name_here \
--region $AWS_REGION
```

Step 2: Check the ECS service discovery logs

The ECS task definition of the CloudWatch agent enables the logs by default in the section below. The logs are sent to CloudWatch Logs in the log group **/ecs/ecs-cwagent-prometheus**.

```
LogConfiguration:
  LogDriver: awslogs
    Options:
      awslogs-create-group: 'True'
      awslogs-group: "/ecs/ecs-cwagent-prometheus"
      awslogs-region: !Ref AWS::Region
      awslogs-stream-prefix: !Sub 'ecs-${ECSLaunchType}-awsvpc'
```

Filter the logs by the string `ECS_SD_Stats` to get the metrics related to the ECS service discovery, as shown in the following example.

```
2020-09-1T01:53:14Z D! ECS_SD_Stats: AWSCLI_DescribeContainerInstances: 1
2020-09-1T01:53:14Z D! ECS_SD_Stats: AWSCLI_DescribeInstancesRequest: 1
2020-09-1T01:53:14Z D! ECS_SD_Stats: AWSCLI_DescribeTaskDefinition: 2
2020-09-1T01:53:14Z D! ECS_SD_Stats: AWSCLI_DescribeTasks: 1
2020-09-1T01:53:14Z D! ECS_SD_Stats: AWSCLI_ListTasks: 1
2020-09-1T01:53:14Z D! ECS_SD_Stats: Exporter_DiscoveredTargetCount: 1
2020-09-1T01:53:14Z D! ECS_SD_Stats: LRUCache_Get_EC2MetaData: 1
2020-09-1T01:53:14Z D! ECS_SD_Stats: LRUCache_Get_TaskDefinition: 2
2020-09-1T01:53:14Z D! ECS_SD_Stats: LRUCache_Size_ContainerInstance: 1
2020-09-1T01:53:14Z D! ECS_SD_Stats: LRUCache_Size_TaskDefinition: 2
2020-09-1T01:53:14Z D! ECS_SD_Stats: Latency: 43.399783ms
```

The meaning of each metric for a particular ECS service discovery cycle is as follows:

- **AWSCLI_DescribeContainerInstances** – the number of `ECS::DescribeContainerInstances` API calls made.
- **AWSCLI_DescribeInstancesRequest** – the number of `ECS::DescribeInstancesRequest` API calls made.
- **AWSCLI_DescribeTaskDefinition** – the number of `ECS::DescribeTaskDefinition` API calls made.
- **AWSCLI_DescribeTasks** – the number of `ECS::DescribeTasks` API calls made.
- **AWSCLI_ListTasks** – the number of `ECS::ListTasks` API calls made.
- **ExporterDiscoveredTargetCount** – the number of Prometheus targets that were discovered and successfully exported into the target result file within the container.
- **LRUCache_Get_EC2MetaData** – the number of times that container instances metadata was retrieved from the cache.
- **LRUCache_Get_TaskDefinition** – the number of times that ECS task definition metadata was retrieved from the cache.
- **LRUCache_Size_ContainerInstance** – the number of unique container instance's metadata cached in memory.
- **LRUCache_Size_TaskDefinition** – the number of unique ECS task definitions cached in memory.
- **Latency** – how long the service discovery cycle takes.

Check the value of `ExporterDiscoveredTargetCount` to see whether the discovered Prometheus targets match your expectations. If not, the possible reasons are as follows:

- The configuration of ECS service discovery might not match your application's setting. For the docker label-based service discovery, your target containers may not have the necessary docker label configured in the CloudWatch agent to auto discover them. For the ECS task definition ARN regular expression-based service discovery, the regex setting in the CloudWatch agent may not match your application's task definition.
- The CloudWatch agent's ECS task role might not have permission to retrieve the metadata of ECS tasks. Check that the CloudWatch agent has been granted the following read-only permissions:
  - `ec2:DescribeInstances`
  - `ecs:ListTasks`
  - `ecs:DescribeContainerInstances`
  - `ecs:DescribeTasks`
  - `ecs:DescribeTaskDefinition`

Step 3: Check the network connection and the ECS task role policy

If there are still no log events sent to the target CloudWatch Logs log group even though the value of `Exporter_DiscoveredTargetCount` indicates that there are discovered Prometheus targets, this could be caused by one of the following:

- The CloudWatch agent might not be able to connect to the Prometheus target ports. Check the security group setting behind the CloudWatch agent. The private IP should alow the CloudWatch agent to connect to the Prometheus exporter ports.
- The CloudWatch agent's ECS task role might not have the **CloudWatchAgentServerPolicy** managed policy. The CloudWatch agent's ECS task role needs to have this policy to be able to send the Prometheus metrics as log events. If you used the sample Amazon CloudFormation template to create the IAM roles automatically, both the ECS task role and the ECS execution role are granted with the least privilege to perform the Prometheus monitoring.

Prometheus metrics troubleshooting on Amazon EKS and Kubernetes clusters

This section provides help for troubleshooting your Prometheus metrics setup on Amazon EKS and Kubernetes clusters.

General troubleshooting steps on Amazon EKS

To confirm that the CloudWatch agent is running, enter the following command.

```
kubectl get pod -n amazon-cloudwatch
```

The output should include a row with `cwagent-prometheus-*id*` in the `NAME` column and `Running` in the `STATUS column.`

To display details about the running pod, enter the following command. Replace *pod-name* with the complete name of your pod that has a name that starts with `cw-agent-prometheus`.

```
kubectl describe pod pod-name -n amazon-cloudwatch
```

If you have CloudWatch Container Insights installed, you can use CloudWatch Logs Insights to query the logs from the CloudWatch agent collecting the Prometheus metrics.

To query the application logs

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **CloudWatch Logs Insights**.
3. Select the log group for the application logs, **/aws/containerinsights/*cluster-name*/application**
4. Replace the search query expression with the following query, and choose **Run query**

```
fields ispresent(kubernetes.pod_name) as haskubernetes_pod_name, stream,
 kubernetes.pod_name, log |
filter haskubernetes_pod_name and kubernetes.pod_name like /cwagent-prometheus
```

You can also confirm that Prometheus metrics and metadata are being ingested as CloudWatch Logs events.

To confirm that Prometheus data is being ingested

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **CloudWatch Logs Insights**.
3. Select the **/aws/containerinsights/*cluster-name*/prometheus**
4. Replace the search query expression with the following query, and choose **Run query**

```
fields @timestamp, @message | sort @timestamp desc | limit 20
```

Logging dropped Prometheus metrics

This release does not collect Prometheus metrics of the histogram type. You can use the CloudWatch agent to check whether any Prometheus metrics are being dropped because they are histogram metrics. You can also log a list of the first 500 Prometheus metrics that are dropped and not sent to CloudWatch because they are histogram metrics.

To see whether any metrics are being dropped, enter the following command:

```
kubectl logs -l "app=cwagent-prometheus" -n amazon-cloudwatch --tail=-1
```

If any metrics are being dropped, you will see the following lines in the `/opt/aws/amazon-cloudwatch-agent/logs/amazon-cloudwatch-agent.log` file.

```
I! Drop Prometheus metrics with unsupported types. Only Gauge, Counter and Summary are
 supported.
I! Please enable CWAgent debug mode to view the first 500 dropped metrics
```

If you see those lines and want to know what metrics are being dropped, use the following steps.

To log a list of dropped Prometheus metrics

1. Change the CloudWatch agent to debug mode by adding the following bold lines to your `prometheus-eks.yaml` or `prometheus-k8s.yaml`file, and save the file.

   ```
   {
         "agent": {
           "debug": true
         },
   ```

   This section of the file should then look like this:

   ```
   cwagentconfig.json: |
       {
         "agent": {
           "debug": true
   ```

```
    },
    "logs": {
      "metrics_collected": {
```

2. Reinstall the CloudWatch agent to enable debug mode by entering the following commands:

```
kubectl delete deployment cwagent-prometheus -n amazon-cloudwatch
kubectl apply -f prometheus.yaml
```

   The dropped metrics are logged in the CloudWatch agent pod.

3. To retrieve the logs from the CloudWatch agent pod, enter the following command:

```
kubectl logs -l "app=cwagent-prometheus" -n amazon-cloudwatch --tail=-1
```

   Or, if you have Container Insights Fluentd logging installed, the logs are also saved in the CloudWatch Logs log group **/aws/containerinsights/*cluster_name*/application**.

   To query these logs, you can follow the steps for querying the application logs in [General troubleshooting steps on Amazon EKS](#).

Where are the Prometheus metrics ingested as CloudWatch Logs log events?

The CloudWatch agent creates a log stream for each Prometheus scrape job configuration. For example, in the `prometheus-eks.yaml` and `prometheus-k8s.yaml` files, the line `job_name: 'kubernetes-pod-appmesh-envoy'` scrapes App Mesh metrics. The Prometheus target is defined as `kubernetes-pod-appmesh-envoy`. So all App Mesh Prometheus metrics are ingested as CloudWatch Logs events in the log stream **kubernetes-pod-appmesh-envoy** under the log group named **/aws/containerinsights/cluster-name/Prometheus**.

I don't see Amazon EKS or Kubernetes Prometheus metrics in CloudWatch metrics

First, make sure that the Prometheus metrics are ingested as log events in the log group **/aws/containerinsights/cluster-name/Prometheus**. Use the information in [Where are the Prometheus metrics ingested as CloudWatch Logs log events?](#) to help you check the target log stream. If the log stream is not created or there are no new log events in the log stream, check the following:

- Check that the Prometheus metrics exporter endpoints are set up correctly
- Check that the Prometheus scraping configurations in the `config map: cwagent-prometheus` section of the CloudWatch agent YAML file is correct. The configuration should

be the same as it would be in a Prometheus configuration file. For more information, see <scrape_config> in the Prometheus documentation.

If the Prometheus metrics are ingested as log events correctly, check that the embedded metric format settings are added into the log events to generate the CloudWatch metrics.

```
"CloudWatchMetrics":[
   {
      "Metrics":[
         {
            "Name":"envoy_http_downstream_cx_destroy_remote_active_rq"
         }
      ],
      "Dimensions":[
         [
            "ClusterName",
            "Namespace"
         ]
      ],
      "Namespace":"ContainerInsights/Prometheus"
   }
],
```

For more information about embedded metric format, see Specification: Embedded metric format.

If there is no embedded metric format in the log events, check that the `metric_declaration` section is configured correctly in the `config map: prometheus-cwagentconfig` section of the CloudWatch agent installation YAML file. For more information, see Tutorial for adding a new Prometheus scrape target: Prometheus API Server metrics.

Integration with Application Insights

Amazon CloudWatch Application Insights helps you monitor your applications and identifies and sets up key metrics, logs, and alarms across your application resources and technology stack. For more information, see Detect common application problems with CloudWatch Application Insights.

You can enable Application Insights to gather additional data from your containerized applications and microservices. If you haven't done this already, you can enable it by choosing **Auto-configure Application Insights** below the performance view in the Container Insights dashboard.

If you have already set up CloudWatch Application Insights to monitor your containerized applications, the Application Insights dashboard appears below the Container Insights dashboard.

For more information about Application Insights and containerized applications, see Enable Application Insights for Amazon ECS and Amazon EKS resource monitoring.

Viewing Amazon ECS lifecycle events within Container Insights

You can view Amazon ECS lifecycle events within the Container Insights console. This helps you correlate your container metrics, logs, and events in a single view to give you a more complete operational visibility.

The events include container instance state change events, task state change events, and service action events. They are automatically sent by Amazon ECS to Amazon EventBridge and are also collected in CloudWatch in event log format. For more information about these events, see Amazon ECS events.

Standard Container Insights pricing applies for Amazon ECS Lifecycle events. For more information, see Amazon CloudWatch Pricing.

To configure the table of lifecycle events and create rules for a cluster, you must have the `events:PutRule`, `events:PutTargets`, and `logs:CreateLogGroup` permissions. You must also make sure that there is a resource policy that enables EventBridge to create the log stream and send logs to CloudWatch Logs. If this resource policy doesn't exist, you can enter the following command to create it:

```
aws --region region logs put-resource-policy --policy-name 'EventBridgeCloudWatchLogs'
 --policy-document '{
    "Statement": [
        {
            "Action": [
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Effect": "Allow",
            "Principal": {
                "Service": ["events.amazonaws.com", "delivery.logs.amazonaws.com"]
            },
            "Resource": "arn:aws:logs:region:account-id:log-group:/aws/events/ecs/
containerinsights/*:*",
            "Sid": "TrustEventBridgeToStoreECSLifecycleLogEvents"
        }
```

```
    ],
    "Version": "2012-10-17"
}'
```

You can use the following command to check whether you already have this policy, and to confirm that attaching it worked correctly.

```
aws logs describe-resource-policies --region region --output json
```

To view the table of lifecycle events, you must have the `events:DescribeRule`, `events:ListTargetsByRule`, and `logs:DescribeLogGroups` permissions.

To view Amazon ECS lifecycle events in the CloudWatch Container Insights console

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. Choose **Insights**, **Container Insights**.
3. Choose **View performance dashboards**.
4. In the next drop-down, choose either **ECS Clusters**, **ECS Services**, or **ECS Tasks**.
5. If you chose **ECS Services** or **ECS Tasks** in the previous step, choose the **Lifecycle events** tab.
6. At the bottom of the page, if you see **Configure lifecycle events**, choose it to create EventBridge rules for your cluster.

   The events are displayed below the container insights panes and above the Application Insights section. To run extra analytics and create additional visualizations on these events, choose **View in Logs Insights** in the Lifecycle Events table.

Troubleshooting Container Insights

The following sections can help if you're having trouble issues with Container Insights.

Failed deployment on Amazon EKS or Kubernetes

If the agent doesn't deploy correctly on a Kubernetes cluster, try the following:

- Run the following command to get the list of pods.

  ```
  kubectl get pods -n amazon-cloudwatch
  ```

- Run the following command and check the events at the bottom of the output.

```
kubectl describe pod pod-name -n amazon-cloudwatch
```

- Run the following command to check the logs.

```
kubectl logs pod-name -n amazon-cloudwatch
```

Unauthorized panic: Cannot retrieve cadvisor data from kubelet

If your deployment fails with the error `Unauthorized panic: Cannot retrieve cadvisor data from kubelet`, your kubelet might not have Webhook authorization mode enabled. This mode is required for Container Insights. For more information, see Verifying prerequisites for Container Insights in CloudWatch.

Deploying Container Insights on a deleted and re-created cluster on Amazon ECS

If you delete an existing Amazon ECS cluster that does not have Container Insights enabled, and you re-create it with the same name, you can't enable Container Insights on this new cluster at the time you re-create it. You can enable it by re-creating it, and then entering the following command:

```
aws ecs update-cluster-settings --cluster myCICluster --settings name=container
 Insights,value=enabled
```

Invalid endpoint error

If you see an error message similar to the following, check to make sure that you replaced all the placeholders such as *cluster-name* and *region-name* in the commands that you are using with the correct information for your deployment.

```
"log": "2020-04-02T08:36:16Z E! cloudwatchlogs: code: InvalidEndpointURL, message:
 invalid endpoint uri, original error: &url.Error{Op:\"parse\", URL:\"https://
logs.{{region_name}}.amazonaws.com/\", Err:\"{\"}, &awserr.baseError{code:
\"InvalidEndpointURL\", message:\"invalid endpoint uri\", errs:[]error{(*url.Error)
(0xc0008723c0)}}\n",
```

Metrics don't appear in the console

If you don't see any Container Insights metrics in the Amazon Web Services Management Console, be sure that you have completed the setup of Container Insights. Metrics don't appear before

Container Insights has been set up completely. For more information, see [Setting up Container Insights](#).

Pod metrics missing on Amazon EKS or Kubernetes after upgrading cluster

This section might be useful if all or some pod metrics are missing after you deploy the CloudWatch agent as a daemonset on a new or upgraded cluster, or you see an error log with the message `W! No pod metric collected`.

These errors can be caused by changes in the container runtime, such as containerd or the docker systemd cgroup driver. You can usually solve this by updating your deployment manifest so that the containerd socket from the host is mounted into the container. See the following example:

```
# For full example see https://github.com/aws-samples/amazon-cloudwatch-container-
insights/blob/latest/k8s-deployment-manifest-templates/deployment-mode/daemonset/
container-insights-monitoring/cwagent/cwagent-daemonset.yaml
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: cloudwatch-agent
  namespace: amazon-cloudwatch
spec:
  template:
    spec:
      containers:
        - name: cloudwatch-agent
# ...
          # Don't change the mountPath
          volumeMounts:
# ...
            - name: dockersock
              mountPath: /var/run/docker.sock
              readOnly: true
            - name: varlibdocker
              mountPath: /var/lib/docker
              readOnly: true
            - name: containerdsock # NEW mount
              mountPath: /run/containerd/containerd.sock
              readOnly: true
# ...
      volumes:
# ...
        - name: dockersock
```

```
          hostPath:
            path: /var/run/docker.sock
        - name: varlibdocker
          hostPath:
            path: /var/lib/docker
        - name: containerdsock # NEW volume
          hostPath:
            path: /run/containerd/containerd.sock
```

No pod metrics when using Bottlerocket for Amazon EKS

Bottlerocket is a Linux-based open source operating system that is purpose-built by Amazon for running containers.

Bottlerocket uses a different `containerd` path on the host, so you need to change the volumes to its location. If you don't, you see an error in the logs that includes `W! No pod metric collected`. See the following example.

```
volumes:
  # ...
    - name: containerdsock
      hostPath:
        # path: /run/containerd/containerd.sock
        # bottlerocket does not mount containerd sock at normal place
        # https://github.com/bottlerocket-os/bottlerocket/
commit/91810c85b83ff4c3660b496e243ef8b55df0973b
        path: /run/dockershim.sock
```

No container filesystem metrics when using the containerd runtime for Amazon EKS or Kubernetes

This is a known issue and is being worked on by community contributors. For more information, see [Disk usage metric for containerd] and [container file system metrics is not supported by cadvisor for containerd] on GitHub.

Unexpected log volume increase from CloudWatch agent when collecting Prometheus metrics

This was a regression introduced in version 1.247347.6b250880 of the CloudWatch agent. This regression has already been fixed in more recent versions of the agent. It's impact was limited to scenarios where customers collected the logs of the CloudWatch agent itself and were also using Prometheus. For more information, see [prometheus] agent is printing all the scraped metrics in log on GitHub.

Latest docker image mentioned in release notes not found from Dockerhub

We update the release note and tag on Github before we start the actual release internally. It usually takes 1-2 weeks to see the latest docker image on registries after we bump the version number on Github. There is no nightly release for the CloudWatch agent container image. You can build the image directly from source at the following location: https://github.com/aws/amazon-cloudwatch-agent/tree/main/amazon-cloudwatch-container-insights/cloudwatch-agent-dockerfile

CrashLoopBackoff error on the CloudWatch agent

If you see a `CrashLoopBackOff` error for the CloudWatch agent, make sure that your IAM permissions are set correctly. For more information, see Verifying prerequisites for Container Insights in CloudWatch.

CloudWatch agent or Fluentd pod stuck in pending

If you have a CloudWatch agent or Fluentd pod stuck in `Pending` or with a `FailedScheduling` error, determine if your nodes have enough compute resources based on the number of cores and amount of RAM required by the agents. Enter the following command to describe the pod:

```
kubectl describe pod cloudwatch-agent-85ppg -n amazon-cloudwatch
```

Building your own CloudWatch agent Docker image

You can build your own CloudWatch agent Docker image by referring to the Dockerfile located at https://github.com/aws-samples/amazon-cloudwatch-container-insights/blob/latest/cloudwatch-agent-dockerfile/Dockerfile.

The Dockerfile supports building multi-architecture images directly using `docker buildx`.

Deploying other CloudWatch agent features in your containers

You can deploy additional monitoring features in your containers using the CloudWatch agent. These features include the following:

- **Embedded Metric Format**— For more information, see Embedding metrics within logs.
- **StatsD**— For more information, see Retrieve custom metrics with StatsD.

Instructions and necessary files are located on GitHub at the following locations:

- For Amazon ECS containers, see Example Amazon ECS task definitions based on deployment modes.
- For Amazon EKS and Kubernetes containers, see Example Kubernetes YAML files based on deployment modes.

CloudWatch Database Insights

Use CloudWatch Database Insights to monitor and troubleshoot Amazon Aurora MySQL, Amazon Aurora PostgreSQL, Amazon RDS for SQL Server, RDS for MySQL, RDS for PostgreSQL, RDS for Oracle, and RDS for MariaDB databases at scale.

With Database Insights, you can monitor your database fleet with pre-built, opinionated dashboards. To help you analyze the performance of your fleet, the Database Insights dashboards display curated metrics and visualizations, and you can customize these dashboards. By presenting metrics in a single dashboard for all databases in your fleet, Database Insights allows you to monitor your databases simultaneously.

For example, you can use Database Insights to find a database that is performing poorly within a fleet of hundreds of database instances. You can then choose that instance and use Database Insights to troubleshoot issues.

For information about engine, Amazon Web Services Region, and instance class support, see Aurora DB engine, Region, and instance class support for Database Insights and Amazon RDS DB engine, Region, and instance class support for Database Insights.

Database Insights supports monitoring workloads only within the same Amazon account.

To get started with Database Insights, see the following topics.

Topics

Modes for Database Insights

Database Insights has an Advanced mode and a Standard mode. Standard mode is the default for Database Insights, and you can turn on the Advanced mode for your database.

The following table shows which features CloudWatch supports for the Advanced mode and Standard mode of Database Insights.

Feature	Standard mode	Advanced mode
Analyze the top contributors to DB Load by dimension	Supported	Supported
Query, graph, and set alarms on database metrics with up to 7 days of retention	Supported	Supported
Define fine-grained access control policies to restrict access to potentially sensitive dimensions such as SQL text	Supported	Supported
Analyze operating system processes happening in your databases with detailed metrics per running process You must have Amazon RDS Enhanced Monitoring enabled to use this feature.	Not supported	Supported
Create and save fleet-wide monitoring views to assess health across hundreds of databases	Not supported	Supported
Analyze SQL locks with 15 months of retention and a guided UX	Not supported	Supported only for Aurora PostgreSQL
Analyze SQL execution plans with 15 months of retention and guided UX	Not supported	Supported only for Aurora PostgreSQL, RDS for Oracle, and RDS for SQL Server
Visualize per-query statistics	Not supported	Supported
Analyze slow SQL queries	Not supported	Supported
View calling services with CloudWatch Application Signals	Not supported	Supported

Feature	Standard mode	Advanced mode
View a consolidated dashboard for all database telemetry, including metrics, logs, events, and applications	Not supported	Supported
Import Performance Insights counter metrics into CloudWatch automatically	Not supported	Supported
View Amazon RDS events in CloudWatch	Not supported	Supported
Analyze database performance for a time period of your choice with on-demand analysis	Not supported	Supported only for Aurora PostgreSQL, Aurora MySQL, RDS for PostgreSQL, RDS for MySQL, and RDS for MariaDB

Note

Database Insights feature availability differs in different Amazon Regions, because not all Advanced Mode features are available in all Regions.

Data retention

The Advanced mode of Database Insights retains 15 months of metrics collected by Performance Insights.

If Performance Insights is enabled for the Standard mode, Amazon RDS retains 7 days of Performance Insights counter metrics.

For information about counter metrics for Performance Insights, see Performance Insights counter metrics.

For information about the retention period for CloudWatch metrics collected by Database Insights, see the following topics.

- Amazon CloudWatch metrics for Amazon Aurora in the *Amazon Aurora User Guide*

- Amazon CloudWatch metrics for Amazon Relational Database Service in the *Amazon RDS User Guide*
- Amazon CloudWatch metrics for Amazon RDS Performance Insights in the *Amazon Aurora User Guide*
- Amazon CloudWatch metrics for Amazon RDS Performance Insights in the *Amazon Aurora User Guide*

How Database Insights integrates with Performance Insights

Performance Insights is a database performance monitoring service.

Database Insights builds upon and extends the capabilities of Performance Insights. Database Insights adds monitoring, analysis, and optimization features.

To enable the Advanced mode of Database Insights, you must enable Performance Insights.

Database Insights imports Performance Insights counter metrics into CloudWatch automatically. The Advanced mode of Database Insights automatically retains 15 months of all metrics collected by Database Insights, including Performance Insights metrics and CloudWatch metrics. This automatically happens for you when you enable Advanced mode in an instance, with no further configuration needed. For information about Performance Insights counter metrics, see Performance Insights counter metrics in the *Amazon Aurora User Guide*.

Pricing

For information about pricing, see Amazon CloudWatch Pricing.

Get started with CloudWatch Database Insights

The Standard mode of Database Insights is enabled by default for your Amazon RDS and Aurora databases. To get started with the Advanced mode of Database Insights, you can create a new database or modify a database.

For information about enabling the Advanced mode or the Standard mode of Database Insights for an Amazon RDS database, see the following topics.

- Turning on the Advanced mode of Database Insights for Amazon RDS in the *Amazon RDS User Guide*

- Turning on the Standard mode of Database Insights for Amazon RDS in the *Amazon RDS User Guide*
- Turning CloudWatch Database Insights on or off when creating a DB instance or Multi-AZ DB cluster for Amazon RDS in the *Amazon RDS User Guide*

For information about enabling the Advanced mode or the Standard mode of Database Insights for an Amazon Aurora database, see the following topics.

- Turning on the Advanced mode of Database Insights for Amazon Aurora in the *Amazon Aurora User Guide*
- Turning on the Standard mode of Database Insights for Amazon Aurora in the *Amazon Aurora User Guide*

Required permissions for Database Insights

Certain IAM permissions are required to use Database Insights. Database Insights requires permissions for CloudWatch, CloudWatch Logs, Amazon RDS, and Amazon RDS Performance Insights. You might not need to provide these permissions to your user or role if you have broader permissions.

The following CloudWatch permissions are required to use Database Insights.

- `cloudwatch:BatchGetServiceLevelIndicatorReport`
- `cloudwatch:DescribeAlarms`
- `cloudwatch:GetDashboard`
- `cloudwatch:GetMetricData`
- `cloudwatch:ListMetrics`
- `cloudwatch:PutDashboard`

The following CloudWatch Logs permissions are required to use Database Insights.

- `logs:DescribeLogGroups`
- `logs:GetQueryResults`
- `logs:StartQuery`
- `logs:StopQuery`

The following Amazon RDS permissions are required to use Database Insights.

- `rds:DescribeDBClusters`
- `rds:DescribeDBInstances`
- `rds:DescribeEvents`

The following Performance Insights permissions are required to use Database Insights.

- `pi:ListAvailableResourceMetrics`
- `pi:ListAvailableResourceDimensions`
- `pi:DescribeDimensionKeys`
- `pi:GetDimensionKeyDetails`
- `pi:GetResourceMetrics`
- `pi:ListPerformanceAnalysisReports`
- `pi:GetResourceMetadata`
- `pi:GetPerformanceAnalysisReport`
- `pi:CreatePerformanceAnalysisReport`
- `pi:DeletePerformanceAnalysisReport`
- `pi:ListTagsForResource`
- `pi:TagResource`
- `pi:UntagResource`

The following sample policy contains the permissions required for full access to Database Insights.

Sample policy for full access

```
{
    "Version": "2012-10-17",
    "Statement": [{
    "Effect" : "Allow",
      "Action" : [
        "cloudwatch:BatchGetServiceLevelIndicatorReport",
        "cloudwatch:DescribeAlarms",
        "cloudwatch:GetMetricStatistics",
        "cloudwatch:GetMetricData",
        "cloudwatch:ListMetrics",
```

```
        "cloudwatch:PutDashboard"
      ],
      "Resource" : "*"
    },
    {
    "Effect" : "Allow",
      "Action" : [
        "logs:DescribeLogGroups",
        "logs:GetQueryResults",
        "logs:StartQuery",
        "logs:StopQuery"
      ],
      "Resource" : "*"
    },
    {
    "Effect" : "Allow",
      "Action" : [
        "pi:DescribeDimensionKeys",
        "pi:GetDimensionKeyDetails",
        "pi:GetResourceMetadata",
        "pi:GetResourceMetrics",
        "pi:ListAvailableResourceDimensions",
        "pi:ListAvailableResourceMetrics",
        "pi:CreatePerformanceAnalysisReport",
        "pi:GetPerformanceAnalysisReport",
        "pi:ListPerformanceAnalysisReports",
        "pi:DeletePerformanceAnalysisReport",
        "pi:TagResource",
        "pi:UntagResource",
        "pi:ListTagsForResource"
      ],
      "Resource" : "arn:aws:pi:*:*:*/rds/*"
    },
    {
    "Effect" : "Allow",
      "Action" : [
        "rds:DescribeDBInstances",
        "rds:DescribeDBClusters",
        "rds:DescribeEvents"
      ],
      "Resource" : "*"
    }
  ]
```

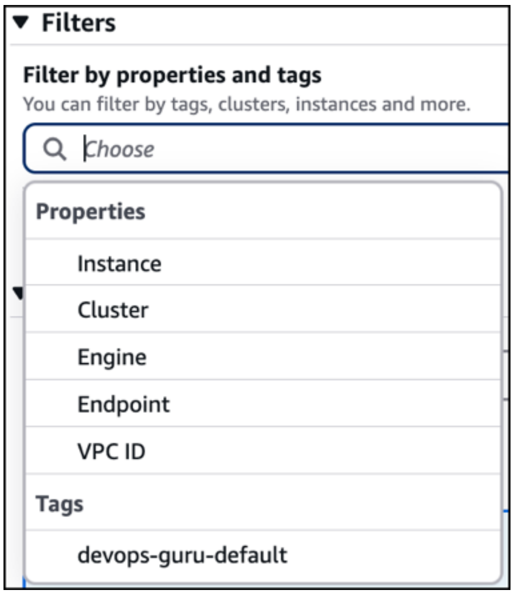
```
}
```

Viewing the Fleet Health Dashboard for CloudWatch Database Insights

You can use the Fleet Health Dashboard to view a snapshot of the health of your database fleet.

Fleet health views

A *database fleet* in Database Insights is a group of databases that you want to monitor. You can create a monitoring view for a database fleet by choosing filters in the **Filters** component. This component allows you to apply filters on properties, such as cluster or instance names and tags. In the Fleet Health Dashboard, CloudWatch shows databases that match at least one of the filter conditions for the fleet health view.

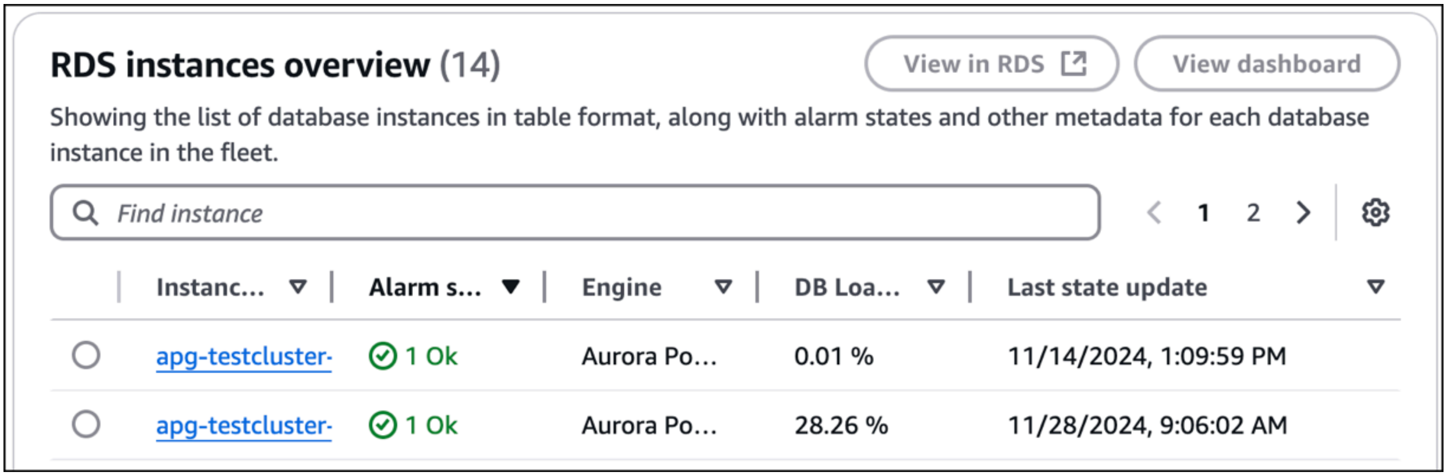


To create, modify, or delete views for database fleets, use the procedures in the following topics.

- Create a fleet health view for CloudWatch Database Insights
- Edit a fleet health view for CloudWatch Database Insights
- Delete a fleet health view for CloudWatch Database Insights

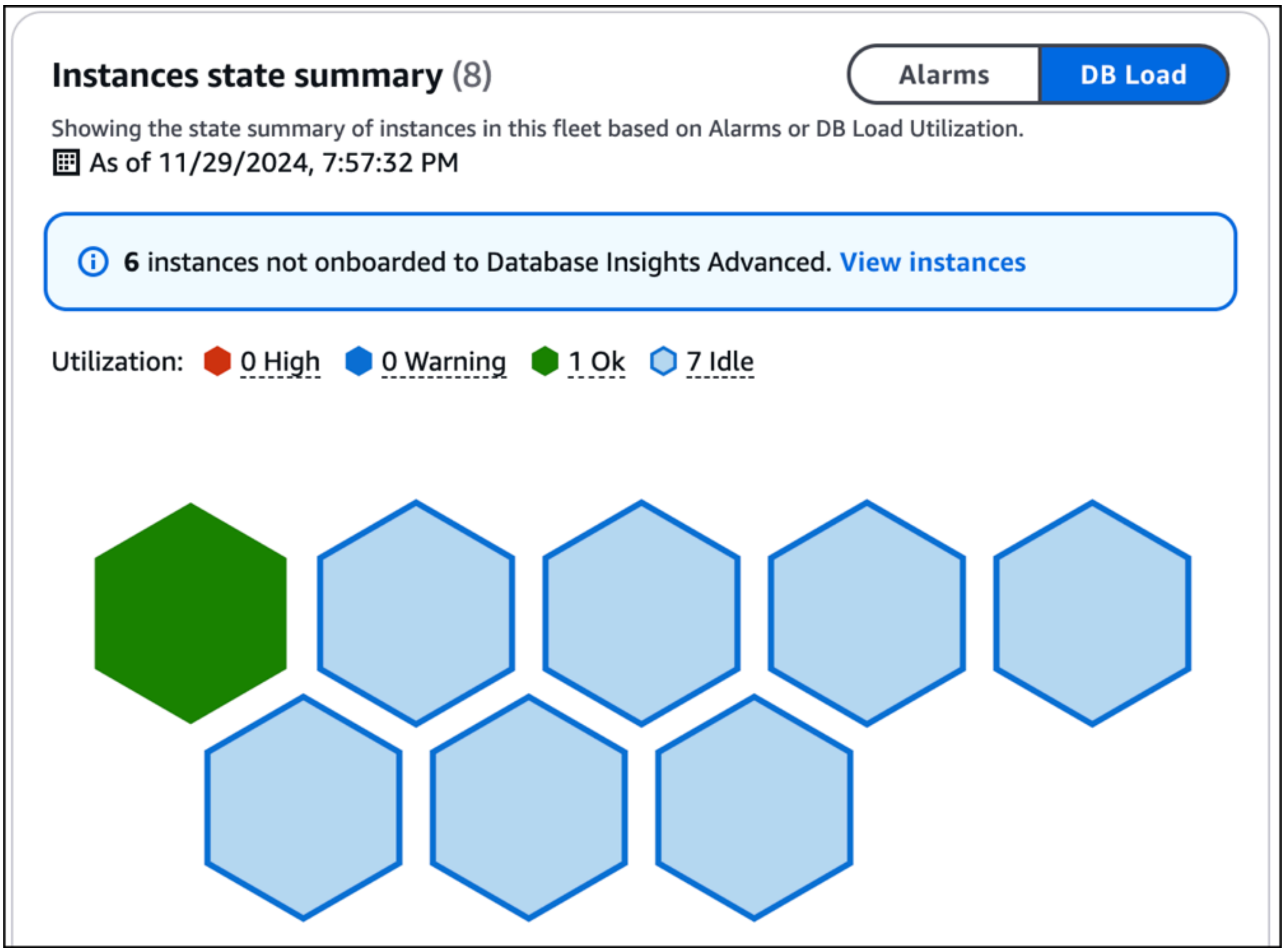
RDS instances overview table

Use the **RDS instances overview** table to view the alarm state, max DB Load percentage, and the time of the last state update for each instance in your fleet.

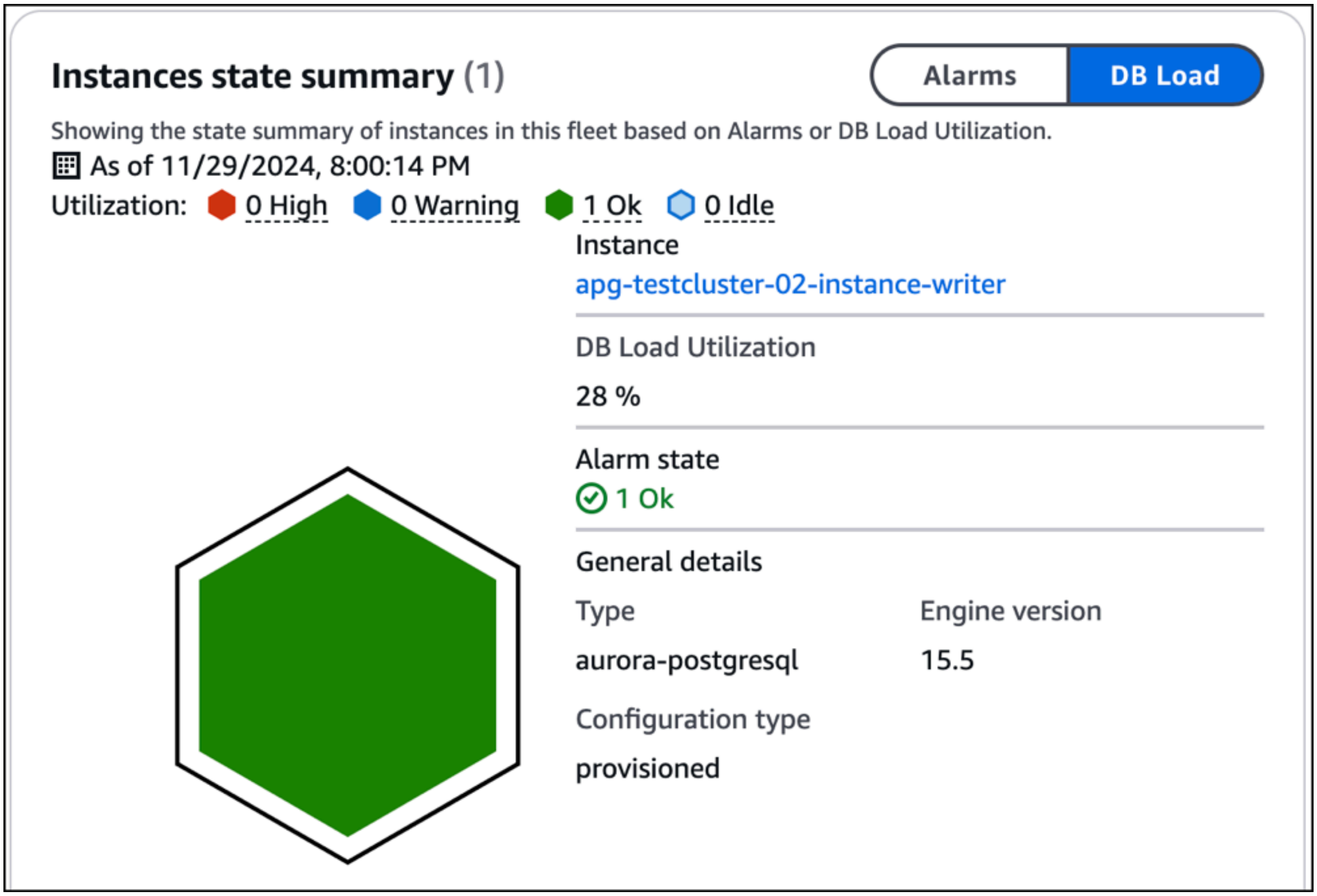


Instances state summary

Use the **Instances state summary** to view the health of all instances in your fleet. The Instances state summary provides two views based on **Alarms** and the DB Load metric. By default, CloudWatch displays the **Alarms** view.



Each node in the honeycomb represents an instance. For more information about an instance, you can choose the corresponding node and choose **Filter view by this instance**.



The honeycomb component summarizes the alarm state for instances in your fleet with the number of nodes in each state at the top of the honeycomb. CloudWatch displays the time of the last refresh of the data shown in the honeycomb.

When you switch to the **DB Load** view, you can see the overall health of the fleet from the point of view of the DB Load metric. Database load (DB Load) measures the number of active sessions in your database. DB Load is the key metric in Database Insights and is collected every second. CloudWatch categorizes DB instances into the following states based on thresholds for DB Load.

- High
- Warning
- Ok
- Idle

You can see the thresholds for DB Load by choosing the corresponding state icons.

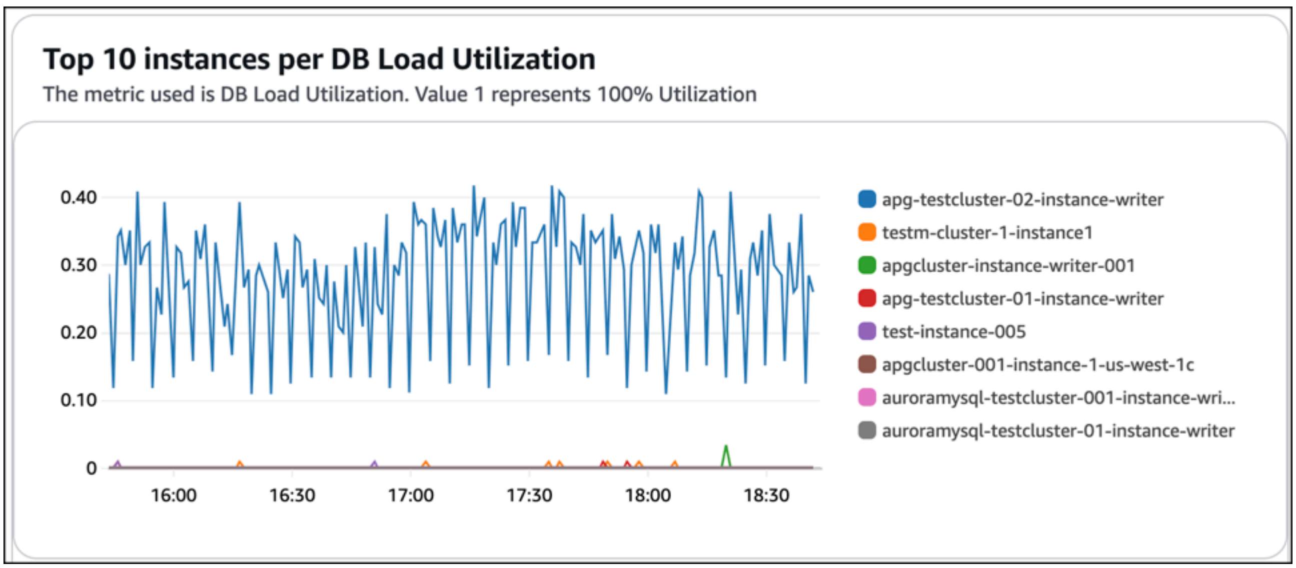
For information about DB Load for Amazon RDS, see Database load in the *Amazon RDS User Guide*. For information about DB Load for Amazon Aurora, see Database load in the *Amazon Aurora User Guide*.

By default, CloudWatch displays the average DB Load. Choose **Max** to monitor the maximum DB Load for each instance.

Choose a node from the Instances state summary to display alarms and DB Load for the instance.

Top 10 charts

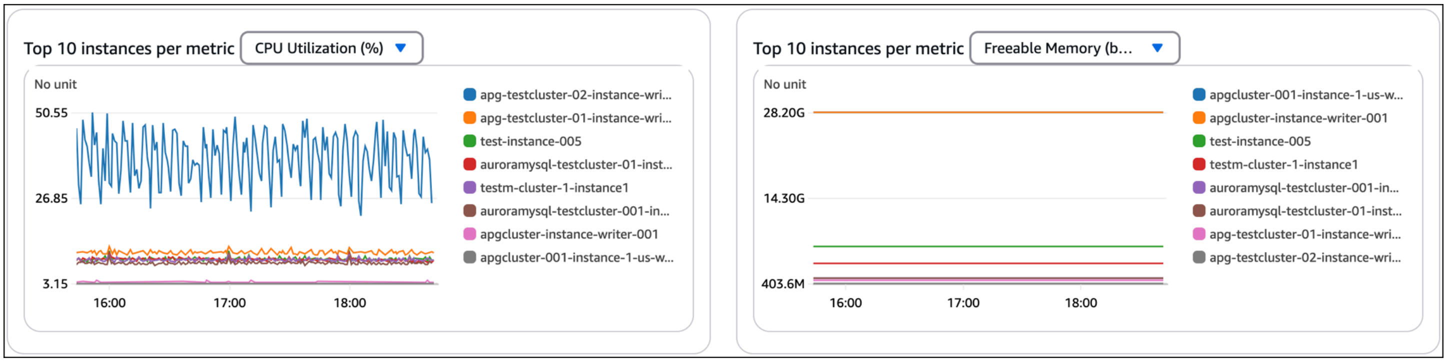
Use the **Top 10 instances per relative DB Load** chart to view the DB Load trend over time for the 10 instances with the highest DB Load. The chart also provides the top queries and top wait events for the instance with the highest DB Load.



Use the **Top 10 instances per metric** charts to compare two key metrics for the top 10 instances in your fleet. You can select the following metrics.

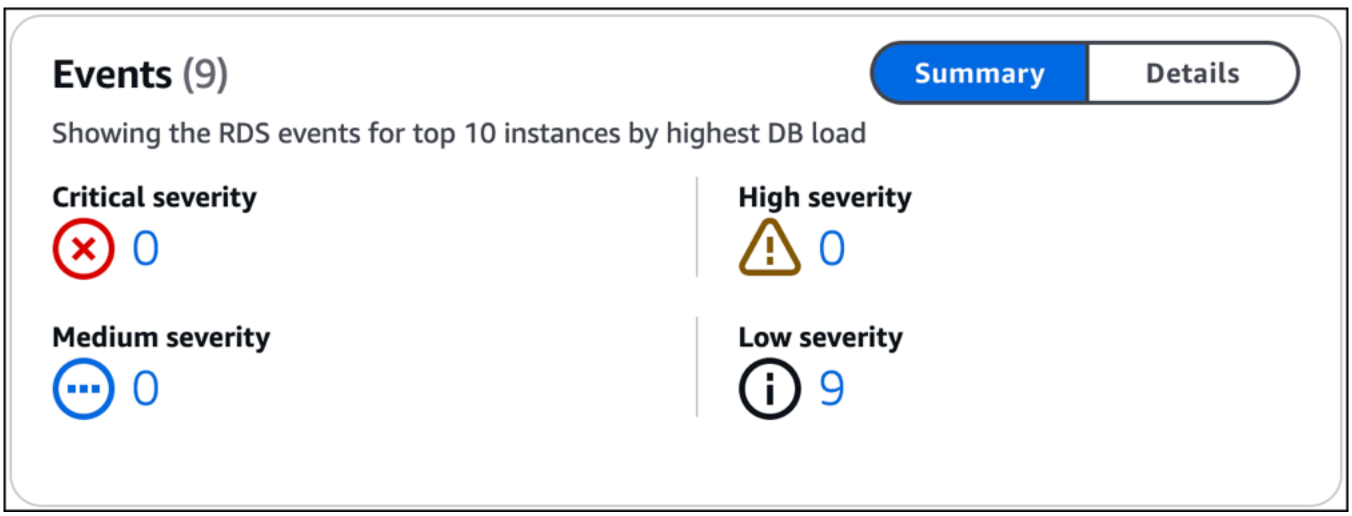
- CPU Utilization (%)
- Freeable Memory (%)
- DB Connections (%)
- Network throughput
- Read IOPS
- Write IOPS

- Read Latency
- Write Latency

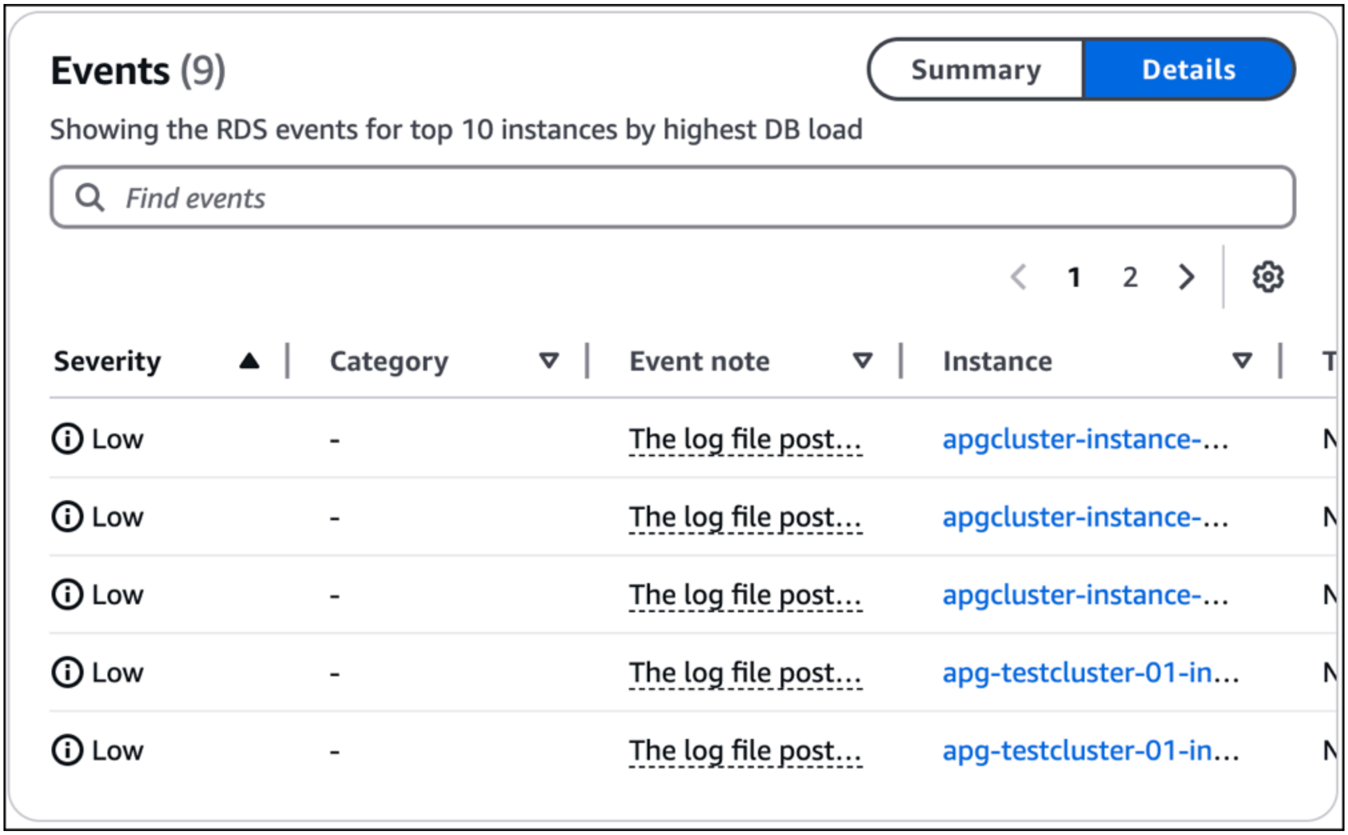


Amazon RDS events

Use the **Events** summary and table to view RDS events for instances in your fleet.



To view the **Events** table, choose **Details**.

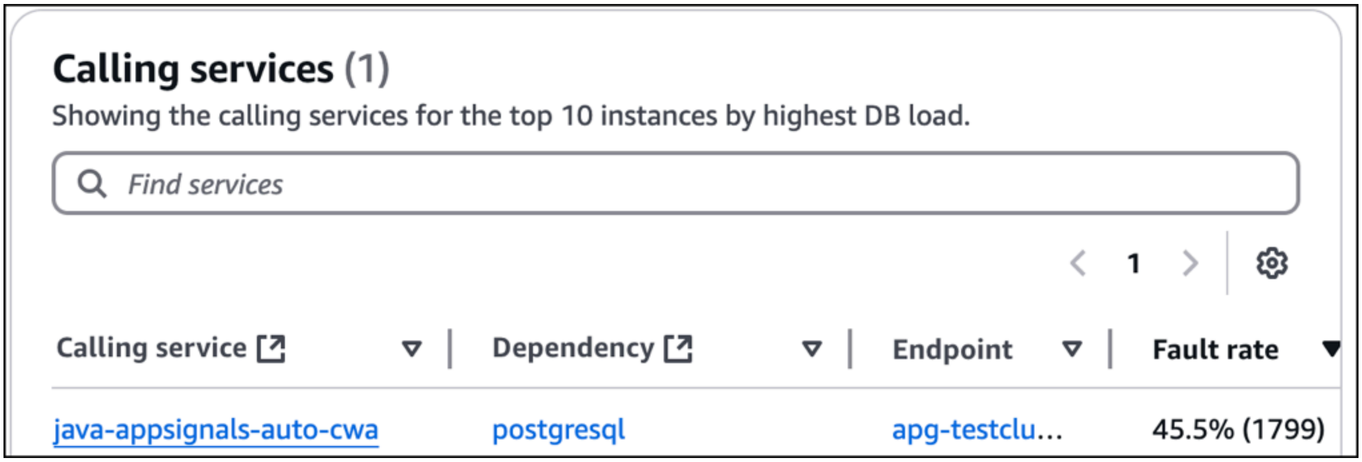


For a list of events for Amazon RDS and Amazon Aurora, see the following topics.

- Amazon RDS event categories and event messages for Aurora in the *Amazon Aurora User Guide*
- Amazon RDS event categories and event messages in the *Amazon RDS User Guide*

Calling services table

Use the **Calling services** table to view CloudWatch Application Signals services that are calling your database endpoints and related application-level metrics such as latency or errors.



Database Insights shows the services that are calling your top 10 instances by DB Load. To view calling services for another instance, choose the instance in the database instance dashboard.

When the endpoint called by the application is an Aurora cluster, Database Insights will display either the writer or the reader endpoint for the Aurora cluster in the **Calling services** table, not the individual database instance. However, when the endpoint called by the application is an Amazon RDS cluster, Database Insights shows the specific database instance the application is calling within the Amazon RDS cluster.

For more information about CloudWatch Application Signals, see [Application Signals](#).

Create a fleet health view for CloudWatch Database Insights

To create a fleet health view, use the following procedure.

To create a fleet health view

1. Sign in to the Amazon Web Services Management Console and open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. Choose **Insights**.
3. Choose **Database Insights**.
4. Choose filters for the fleet of databases you want to monitor.
5. Choose **Save filter as fleet**.
6. In the **Save filter set (fleet)** window, enter a name for your fleet.
7. Choose the **Save** button.

To access the saved fleet, choose the **Saved fleets** dropdown. Then, choose a fleet.

Edit a fleet health view for CloudWatch Database Insights

To edit a fleet health view, use the following procedure.

To edit a fleet health view

1. Sign in to the Amazon Web Services Management Console and open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. Choose **Insights**.
3. Choose **Database Insights**.
4. Choose the **Saved fleets** dropdown.
5. Choose the vertical ellipsis for a fleet health view you want to edit.
6. In the **Edit filter set (fleet)** window, you can edit the name of the fleet and the filters for the fleet.
7. Choose the **Save** button.

Delete a fleet health view for CloudWatch Database Insights

To delete a fleet health view, use the following procedure.

To delete a fleet health view

1. Sign in to the Amazon Web Services Management Console and open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. Choose **Insights**.
3. Choose **Database Insights**.
4. Choose the **Saved fleets** dropdown.
5. Choose the vertical ellipsis for a fleet health view you want to delete.
6. In the **Delete filter set (fleet)** window, choose the **Delete** button.

Viewing the Database Instance Dashboard for CloudWatch Database Insights

Use the Database Instance Dashboard to view a snapshot of the health of a DB instance.

To analyze lock trees and execution plans for Amazon Aurora PostgreSQL, see the following topics.

Topics

- Analyzing lock trees for Amazon Aurora PostgreSQL with CloudWatch Database Insights
- Analyzing execution plans with CloudWatch Database Insights

Database load chart

Database load (DB Load) measures the level of session activity in your database. DB Load is the key metric in Database Insights, and Database Insights collects DB Load every second.

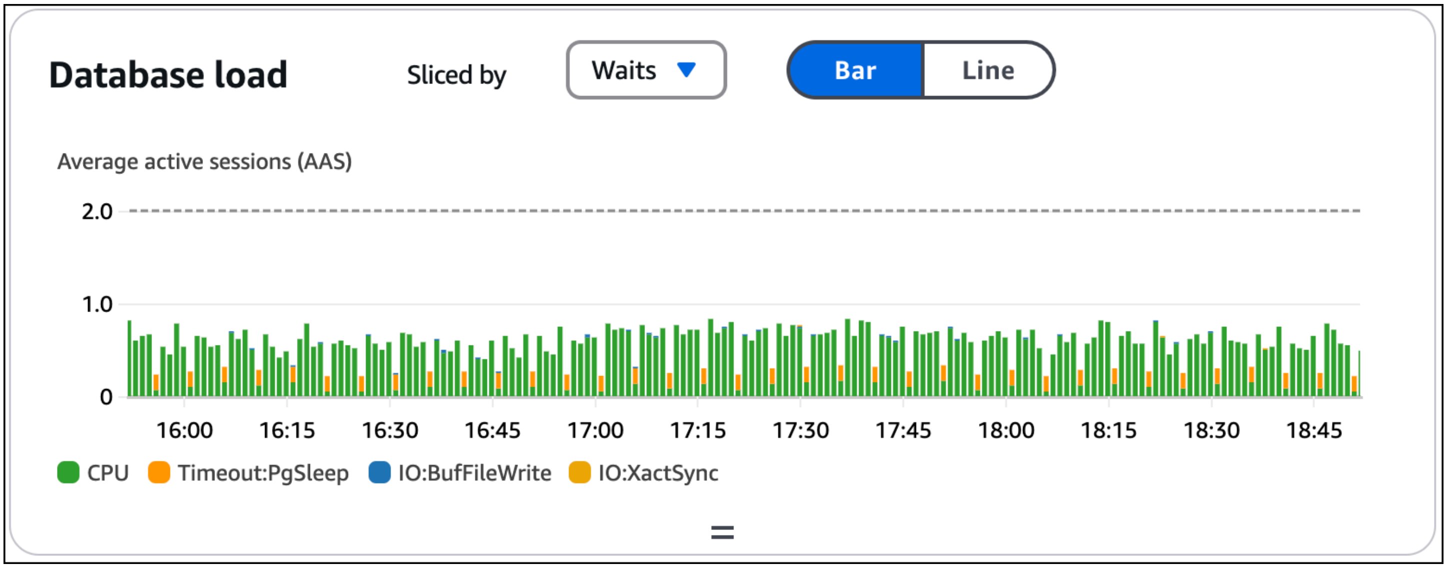
For more information about DB Load, see Database load in the *Amazon RDS User Guide* or Database load in the *Amazon Aurora User Guide*.

Use the **Database load** chart to view DB Load sliced (grouped) by the following dimensions for all supported database engines.

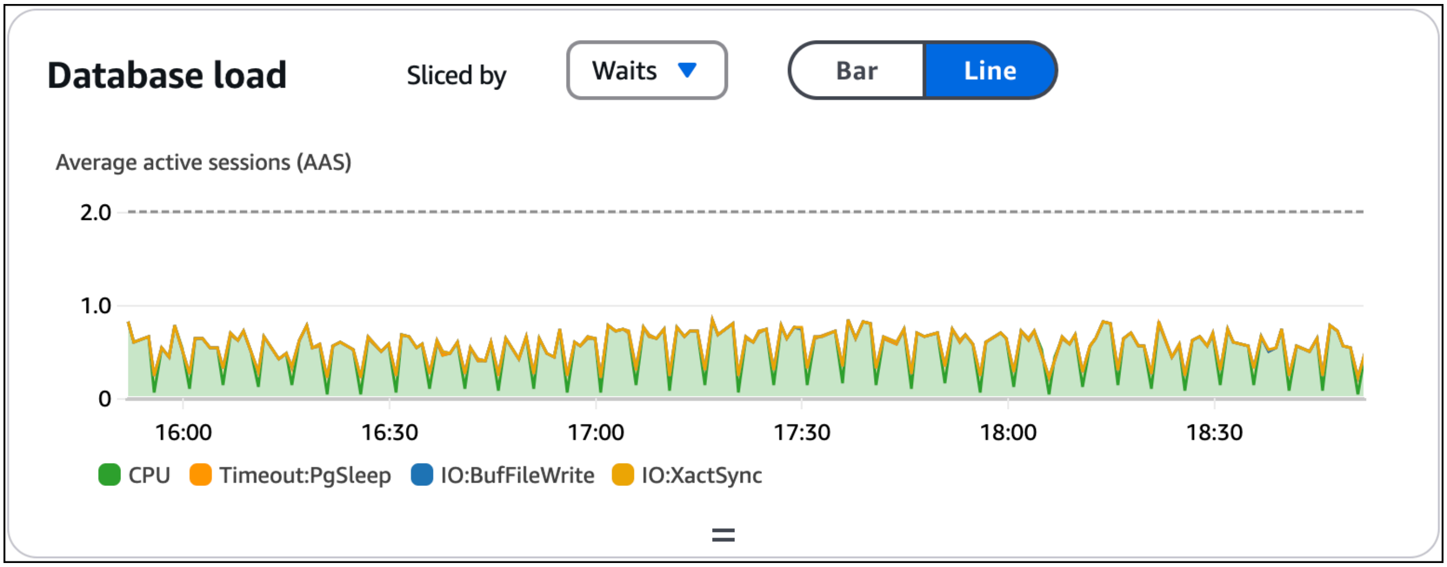
- Blocking object (only for database engines that support locking analysis)
- Blocking session (only for database engines that support locking analysis)
- Blocking SQL (only for database engines that support locking analysis)
- Database
- Host
- SQL
- User
- Waits
- Application (only for Amazon Aurora PostgreSQL)
- Plans (only for database engines that support execution plan capture)
- Session type (only for Amazon Aurora PostgreSQL)

Note

For information about analyzing Oracle PDB load in Amazon RDS, see Analyzing top Oracle PDB load in the *Amazon RDS User Guide*.



By default, CloudWatch displays DB Load with a bar chart. Choose **Line** to display DB Load with a stacked line chart.



DB Load analysis tab

Use the **DB Load analysis** tab to monitor the top contributors to DB Load for each of the following dimensions.

- Database
- Host
- SQL
- User

- Waits
- Lock analysis (only for database engines that support locking analysis)
- Application (only for Amazon Aurora PostgreSQL)
- Session type (only for Amazon Aurora PostgreSQL)

Analyze statistics for a query

You might want to analyze statistics for a query with a high DB Load. To analyze statistics for a query, use the following procedure.

To analyze statistics for queries

1. Sign in to the Amazon Web Services Management Console and open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. Choose **Insights**.
3. Choose **Database Insights**.
4. Choose the **Database Instance** view.
5. Choose a DB instance.
6. Choose the **Top SQL** tab.
7. To view statistics for a query, choose a query.

Database telemetry tab

Use the **Database telemetry** tab to view metrics, logs, events, and slow queries for the selected instance.

Metrics section for database telemetry

The **Metrics** section displays a default metrics dashboard customized for each engine type.

You can customize this dashboard by adding OS metrics, database counter metrics, and CloudWatch metrics to it. You can also remove metrics from the dashboard. You can customize one dashboard for each engine type in a Region in your account. This means that all instances for a specific engine type in that Region in the same account will have the same metrics dashboard.

Users who have edit permissions for your dashboards in your account can edit any dashboard for any engine.

Changes you make to a dashboard are saved automatically, and apply to every instance of the database engine in that Region and account.

To customize the dashboard in the Database telemetry tab for an engine type

1. Sign in to the Amazon Web Services Management Console and open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. Choose **Insights**, **Database Insights**.
3. For **Database Views**, choose **Database Instance**.
4. In the **Filters** section, find and choose the database instance that you want to view metrics for.
5. Choose the **Database Telemetry** tab, then choose the **Metrics** tab.

   The default database instance dashboard appears.
6. To add a widget to the dashboard, do the following:

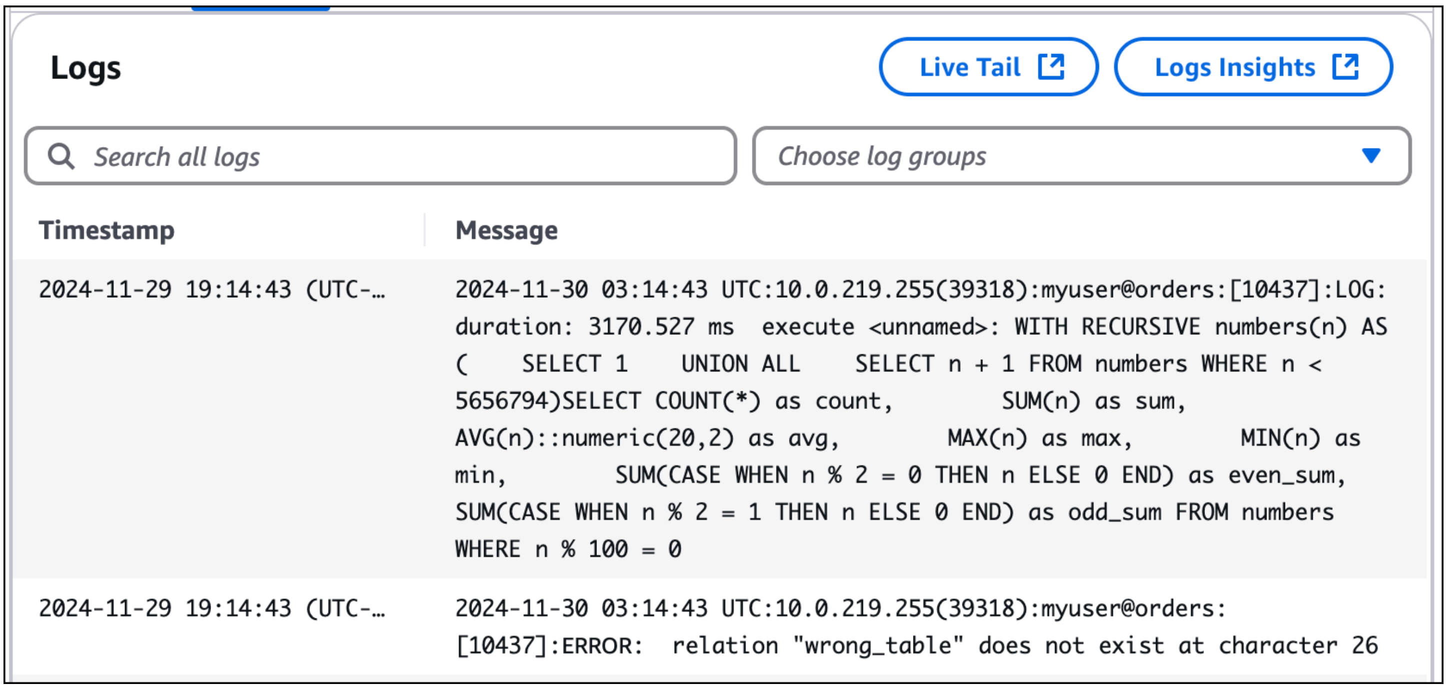
   a. Choose **Add Create widget**.
   b. In the **Create widget** popup, find the metric or metrics that you want to add, and select the checkbox for each one. If you select multiple metrics in this step, they will all appear in the same new widget on the dashboard. Then choose **Create widget**.

      Remember that any changes you make to this dashboard will apply to all Database Insights dashboards for this engine type in this Region in the account.

7. To delete a graph from the dashboard, choose the vertical ellipsis in the widget, then choose **Delete**.
8. To add more metrics to an existing widget in the dashboard, or change its title, choose the vertical ellipsis in the widget, and choose **Edit**. Then in the **Update widget** popup, find the metric or metrics that you want to add, select their checkboxes, and choose **Update Widget**. You can also change the widget title.
9. After customizing a dashboard, you can reset it to its original default state by choosing **Reset Dashboard**.

Logs section for database telemetry

The **Logs** section provides a view of database logs exported to CloudWatch Logs for the selected DB instance.



For information about publishing logs to CloudWatch Logs for Amazon RDS, see Publishing database logs to Amazon CloudWatch Logs in the *Amazon RDS User Guide*. For information about publishing logs to CloudWatch Logs for Amazon Aurora, see Publishing database logs to Amazon CloudWatch Logs in the *Amazon Aurora User Guide*.

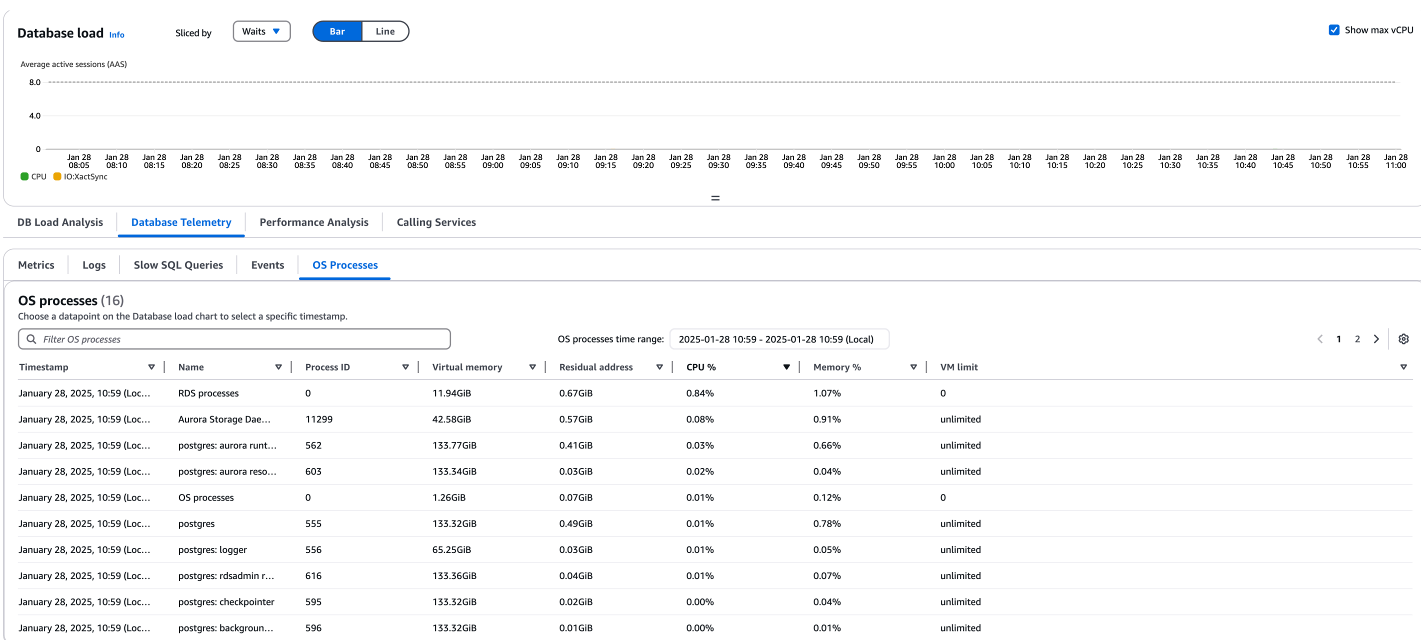
Amazon RDS and OS processes data for database telemetry

You can use the **OS Processes** tab within the **Database telemetry** tab to view metrics for the operating system (OS) that your DB instance runs on. The metrics provide a snapshot of OS processes running on your databases for a given timestamp, as well as key metrics such as memory and CPU utilization for each running process. Database Insights correlates these metrics with the metrics in your database load chart, so if you choose a data point in the database load chart, the OS processes data is updated to display telemetry from the same time stamp.

When you choose a data point, Database Insights automatically select the period to display, depending on the time range you have chosen for the overall page. The farthest back that you can go depends on the retention time that you have configured for the `RDSOSMetrics` log group.

If you haven't chosen a time stamp, by default the table is populated with telemetry for the latest timestamp.

> **Note**
>
> OS process information is available only if you have [Amazon RDS Enhanced Monitoring](#) enabled. Enhanced Monitoring incurs additional charges. For more information, see [Cost of Enhanced Monitoring](#).



In the **OS processes** view, the following data is displayed for each process:

- **Process ID** –The ID of this process.
- **Virtual memory** –The amount of virtual memory allocated to the process, in Kibibytes.

- **Residual address –** The actual physical memory being used by the process.
- **CPU % –** The percentage of the total CPU bandwidth being used by the process.
- **Memory % –** The percentage of the total memory being used by the process.
- **VM limit –** The maximum amount of virtual memory that can be allocated to the process.

  If the value in this column is 0, then VM limits are not applicable to that process.

The monitoring data that is displayed is retrieved from Amazon CloudWatch Logs. You can also retrieve these metrics directly from the log stream in CloudWatch Logs. For more information, see Viewing OS metrics using CloudWatch Logs.

OS processes metrics are not returned during the following:

- A failover of the database instance.
- Changing the instance class of the database instance (scale compute).

OS processes metrics are returned during a reboot of a database instance because only the database engine is rebooted. Metrics for the operating system are still reported.

Slow SQL Queries section for database telemetry

To view slow SQL queries and query patterns, you must enable log exports to CloudWatch Logs and configure DB parameters for your database.

For information about publishing Amazon RDS logs to CloudWatch Logs, see Publishing database logs to Amazon CloudWatch Logs in the *Amazon RDS User Guide*.

For information about publishing Aurora logs to CloudWatch Logs, see Publishing database logs to Amazon CloudWatch Logs in the *Amazon Aurora User Guide*.

For information about configuring DB parameters for your database in Amazon RDS, see Configuring your database to monitor slow SQL queries with Database Insights for Amazon RDS in the *Amazon RDS User Guide*.

For information about configuring DB parameters for your database in Amazon Aurora, see Configuring your database to monitor slow SQL queries with Database Insights for Amazon Aurora in the *Amazon Aurora User Guide*.

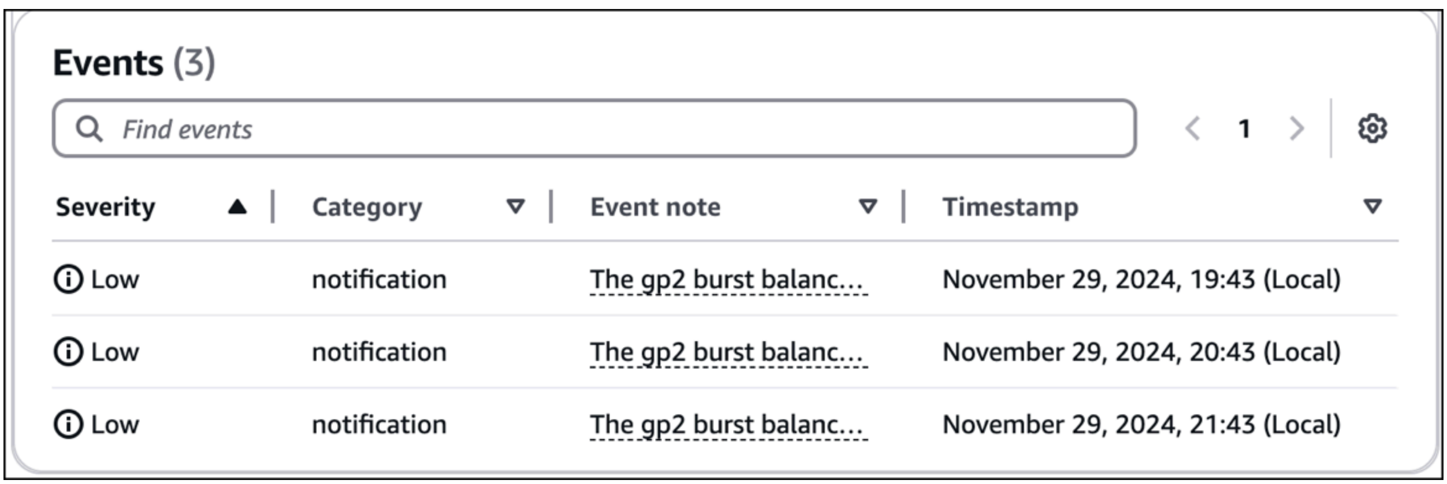
The **Slow SQL Queries** section provides a list of slow query patterns sorted by frequency. By selecting a pattern, you can view a list of slow queries that match the selected pattern. You can use the slow query list to identify slow queries affecting your DB instance.

Database Insights displays statistics for slow queries. The statistics represent only queries that exceed the configured slow query duration threshold.

> **Important**
>
> Slow queries may contain sensitive data. Mask your sensitive data with CloudWatch Logs. For more information about masking log data, see Help protect sensitive log data with masking in the *Amazon CloudWatch Logs User Guide*.

Events table

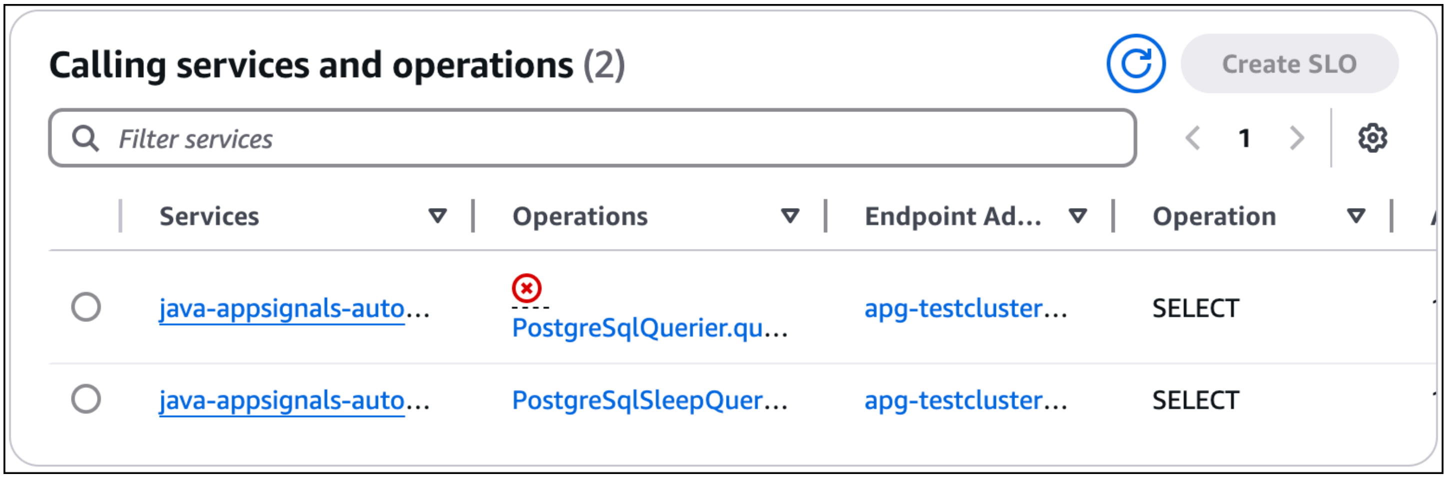
Use the **Events** table to view RDS events for your DB instance. For a list of events for Amazon Aurora, see Amazon RDS event categories and event messages for Aurora in the *Amazon Aurora User Guide*. For a list of events for Amazon Relational Database Service, see Amazon RDS event categories and event messages for Aurora in the *Amazon RDS User Guide*.



Calling services tab

Database Insights shows the services and operations that are calling your instance. Database Insights integrates with CloudWatch Application Signals to provide metrics for each service and operation, including availability, latency, errors, and volume.

When the endpoint called by the application is an Aurora cluster, Database Insights will show either the writer or the reader endpoint for the Aurora cluster in the **Calling services** table, not the individual database instance. However, when the endpoint called by the application is an Amazon RDS cluster, Database Insights shows the specific database instance the application is calling within the Amazon RDS cluster."



Analyze database performance on demand with CloudWatch Database Insights

Analyze database performance with on-demand analyses for your Amazon RDS databases with CloudWatch Database Insights.

Use the **Performance analysis** tab to view performance analysis reports for databases in your fleet.

For information about performance analysis reports for Amazon Aurora, see Analyzing database performance for a period of time in the *Amazon Aurora User Guide*.

> **Note**
>
> You must create performance analysis reports using the Amazon RDS console, Amazon CLI, or API. For information about creating a performance analysis report for Amazon Aurora, see Creating a performance analysis report in Performance Insights in the *Amazon Aurora User Guide*.

Integrating CloudWatch Database Insights with CloudWatch Application Signals

Integrate CloudWatch Database Insights with CloudWatch Application Signals.

Use the **Calling services** tab to view the CloudWatch Application Signals services and operations that called an endpoint of the selected instance. By default, CloudWatch sorts the table by fault rate. Choose values in the **Services**, **Operations**, or **Endpoint address** columns to view the corresponding resource in the CloudWatch Application Signals console.

For more information about supported systems for CloudWatch Application Signals, see Supported systems.

Analyzing lock trees for Amazon Aurora PostgreSQL with CloudWatch Database Insights

To troubleshoot performance issues caused by locks, you can analyze lock trees for Amazon Aurora PostgreSQL databases with CloudWatch Database Insights using the following.

- **Sliced by** dropdown – Choose the **Blocking object**, **Blocking session**, or **Blocking SQL** dimensions in the **Database load** chart to view how distinct top blockers contribute to DB Load over time. With the DB load chart, you can analyze if top blockers are constant or change often. Then, you can troubleshoot the blockers.

- **Lock analysis** tab – Choose **DB Load Analysis**, then choose the **Lock analysis** tab to view information about lock contention in your database.

> **Note**
>
> CloudWatch Database Insights supports lock analysis for all Aurora PostgreSQL versions. To analyze lock trees, you must have Database Insights Advance Mode enabled. For information on how to turn on Advanced mode, see Turning on the Advanced mode of Database Insights for Amazon Aurora and Turning on the Advanced mode of Database Insights for Amazon Relational Database Service

The lock analysis tab provides information about lock contention for your database. The lock tree visualization shows the relationships and dependencies between lock requests from different sessions.

Database Insights captures snapshots every 15 seconds. Snapshots show the lock data for your database at a point in time.

> **Note**
>
> When CloudWatch detects high locking, CloudWatch displays the **High locking detected** banner for the **Lock analysis** tab. CloudWatch detects high locking if CloudWatch takes a lock snapshot for each 15 second interval for 15 consecutive minutes.

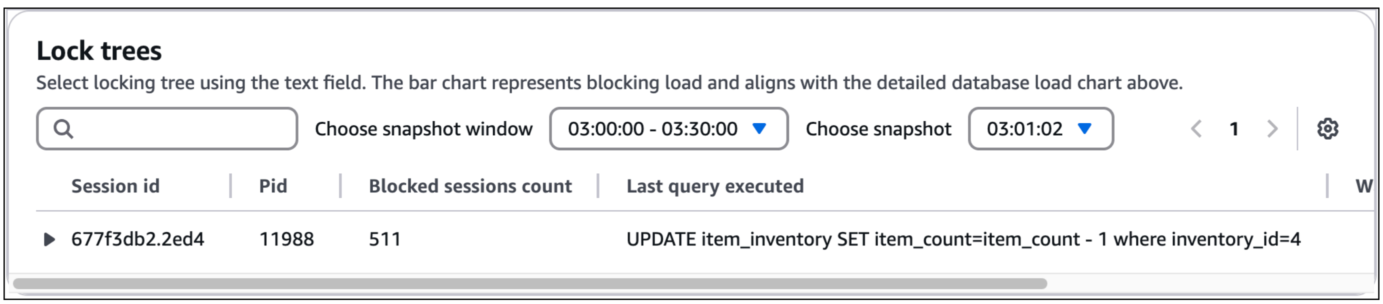
Each node in the tree represents a specific session. The parent node is a session that is blocking its child nodes.

To analyze lock trees, use the following procedure.

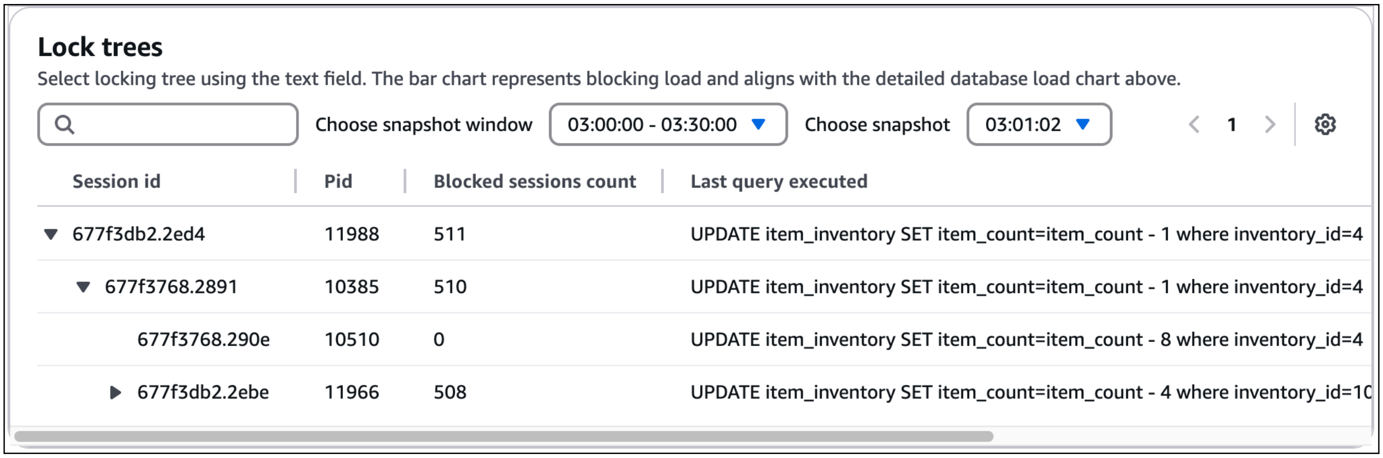
To analyze lock trees

1. Sign in to the Amazon Web Services Management Console and open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. Choose **Insights**.
3. Choose **Database Insights**.
4. Choose the **Database Instance** view.
5. Choose a DB instance.
6. Choose the **DB load analysis** tab.
7. Choose the **Lock analysis** tab.

   To view lock data for a DB instance, choose a period of 1 day or less.
8. Choose a snapshot window. By default, Database Insights chooses the snapshot window with the most blocked sessions.

9. To view lock data for a snapshot, choose the time Database Insights took the snapshot.
10. To expand a lock tree, choose the arrow next to the session ID.



Lock snapshot data

Database Insights provides the following information for each lock request. To view columns that aren't enabled by default, choose the **Settings** icon for the **Lock trees** table and enable other columns.

Column name	Definition	Default column	Notes
`session_id`	The unique session identifier.	Yes	The `session_id` is derived from `HEX(pg_stat_activity.backend_start).HEX(pg_locks.pid)`.
`pid`	The PID of this backend.	Yes	`pg_locks.pid`
`blocked_sessions_count`	The number of sessions blocked by this lock.	Yes	The `blocked_sessions_count` is derived from the number of session IDs blocked by this lock.

Column name	Definition	Default column	Notes
`last_query_executed`	The last query executed by this session. For blockers, it may not be the query that holds the blocking lock.	Yes	`pg_stat_activity.query`
`wait_event`	The wait event name if the backend is currently waiting, otherwise the value is NULL.	Yes	`pg_stat_activity.wait_event`
`blocking_time_(In Seconds)`	The time (in seconds) since the start of this lock.	Yes	The `blocking_time_(In Seconds)` is derived from the start time of the waiting transaction (`pg_locks.waitstart`) for the first waiter.
`blocking_mode`	The lock mode held by the blocking session.	No	`pg_locks.mode`
`waiting_mode`	The lock mode requested by the waiting session.	No	`pg_locks.mode`
`application`	The name of the application that is connected to this backend.	No	`pg_stat_activity.application_name`
`blocking_txn_start_time`	The start time of the blocking transaction or null if no transaction is active.	No	`pg_stat_activity.xact_start`
`waiting_start_time`	The time when a waiting user session started waiting for this lock, or null if the lock is held.	No	`pg_locks.waitstart`

Column name	Definition	Default column	Notes
session_start_time	The time when a user session was started.	No	pg_stat_activity.backend_start
state	The state of a backend.	No	pg_stat_activity.state
wait_event_type	The type of wait event for which this session is waiting.	No	pg_stat_activity.wait_event_type
last_query_exec_time	The time when the last query was started.	No	pg_stat_activity.query_start
user	The name of the user logged into this backend.	No	pg_stat_activity.usename
host	The host name of the connected client, as reported by a reverse DNS lookup of `client_addr` . This field will only be non-null for IP connections, and only when [log_hostname](#) is enabled.	No	pg_stat_activity.client_hostname
port	The TCP port number that the client is using for communication with this backend, or `-1` if a Unix socket is used. If this field is null, it indicates that this is an internal server process.	No	pg_stat_activity.client_port

Column name	Definition	Default column	Notes
client_address	The IP address of the client connected to this backend. If this field is null, it indicates either that the client is connected via a Unix socket on the server machine or that this is an internal process such as autovacuum.	No	pg_stat_activity.client_addr
granted	The value is true if lock is held and false if lock is awaited.	No	pg_locks.granted
waiting_tuple	The tuple number targeted by the lock within the page, or null if the target is not a tuple.	No	pg_locks.tuple
waiting_page	The page number targeted by the lock within the relation, or null if the target is not a relation page or tuple.	No	pg_locks.page
waiting_transaction_id	The ID of the transaction targeted by the lock, or null if the target is not a transaction ID.	No	pg_locks.transactionid
waiting_relation	The OID of the relation targeted by the lock, or null if the target is not a relation or part of a relation.	No	pg_locks.relation
waiting_object_id	The OID of the lock target within its system catalog, or null if the target is not a general database object.	No	pg_locks.objid
waiting_database_id	The OID of the database in which the lock target exists, or zero if the target is a shared object, or null if the target is a transaction ID.	No	pg_locks.database

Column name	Definition	Default column	Notes
waiting_database_name	The name of the database in which the lock target exists.	No	pg_stat_activity.datname
waiting_locktype	The type of the lockable object: relation, extend, frozenid, page, tuple, transactionid, virtualxid, spectoken, object, userlock, advisory, or applytransaction.	No	pg_locks.locktype
is_fastpath	The value is true if the lock was taken with the fast path and false if taken from the main lock table.	No	pg_locks.fastpath

For more information about the values in the `pg_stat_activity` and `pg_locks` views, see the following topics in the PostgreSQL documentation.

- pg_locks
- pg_stat_activity

Analyzing execution plans with CloudWatch Database Insights

You can analyze execution plans for the Amazon Aurora PostgreSQL, RDS for Microsoft SQL Server and RDS for Oracle databases by using the following methods.

- **Sliced by** dropdown – Choose the **Plans** dimension in the **Database load** chart to view how different plans contribute to DB Load over time.
- **Top SQL** tab – Choose **DB Load Analysis**, then choose the **Top SQL** tab to view the number of plans for each digest query.

  To analyze execution plans for a digest query, choose the query and then choose the **Plans** tab. For more information, see the following procedure.

Prerequisites

To analyze execution plans, you must be using the Advanced mode of Database Insights. For information on how to turn on Advanced mode, see Turning on the Advanced mode of Database Insights for Amazon Aurora and Turning on the Advanced mode of Database Insights for Amazon Relational Database Service.

If you are using Aurora PostgreSQL, you also have the following prerequisites:

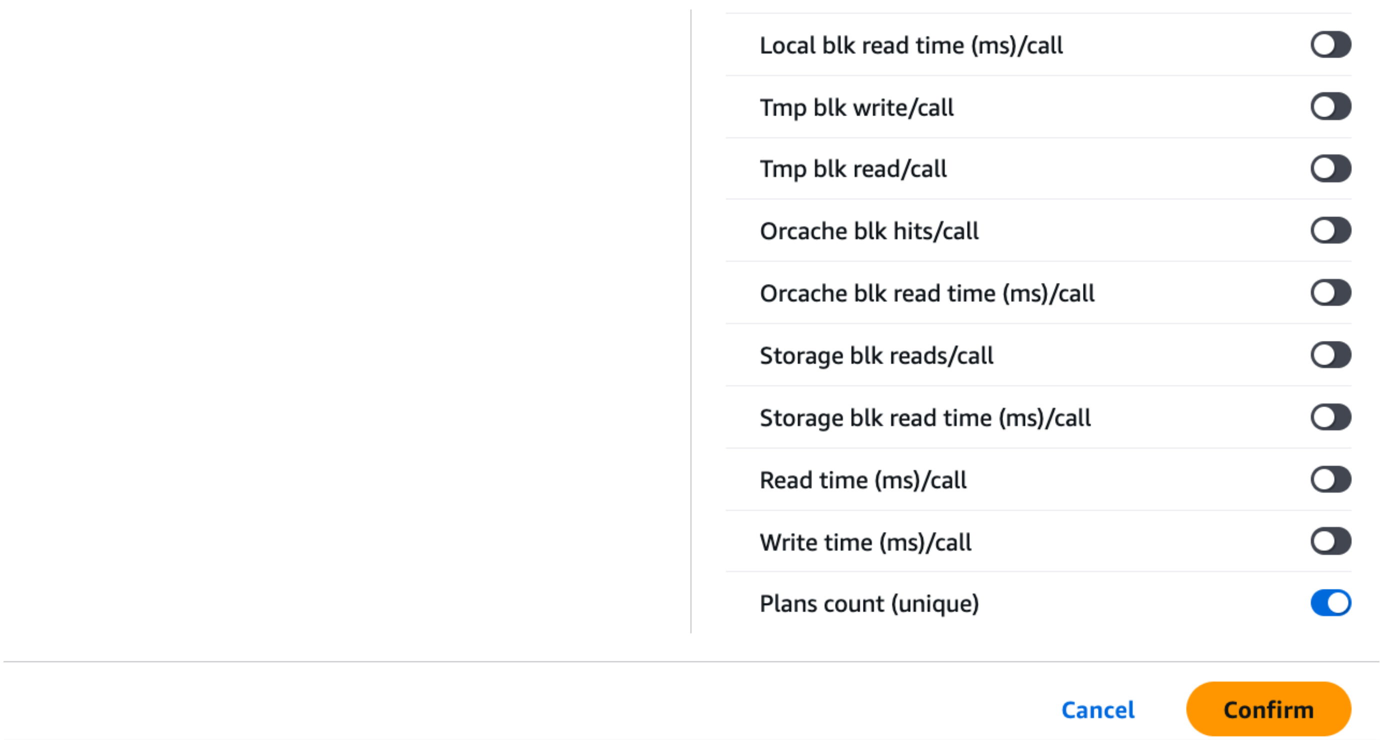
- Your DB instance must use Aurora PostgreSQL version 14.10, 15.5, or later. For information about upgrading your Aurora PostgreSQL DB cluster, see Upgrading Amazon Aurora PostgreSQL DB clusters in the *Amazon Aurora User Guide*.
- You must configure your DB cluster to analyze execution plans by setting the parameter `aurora_compute_plan_id` to on with one of the following options.
  - Creating a DB cluster parameter group in Amazon Aurora in the *Amazon Aurora User Guide*
  - Modifying parameters in a DB cluster parameter group in Amazon Aurora in the *Amazon Aurora User Guide*

Analyze execution plans

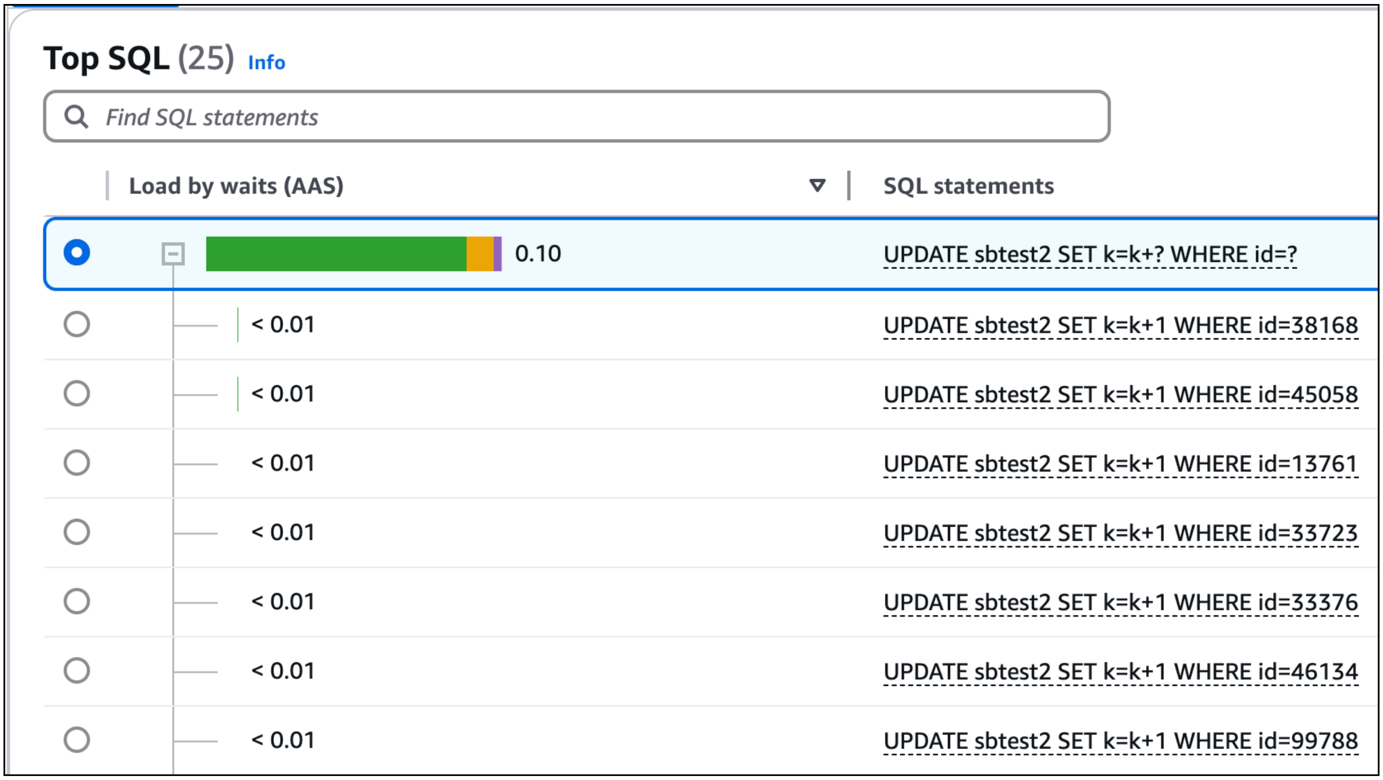
To analyze execution plans, use the following procedure.

To analyze execution plans

1. Sign in to the Amazon Web Services Management Console and open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. Choose **Insights**.
3. Choose **Database Insights**.
4. Choose the **Database Instance** view.
5. Choose a DB instance.
6. Choose the **Top SQL** tab. The **Plans Count** column shows the number of plans collected for each digest query.
7. (Optional) If the **Plans Count** column doesn't appear, choose the **Settings** icon on the **Top SQL** table to customize the visibility and order of columns.



8. Choose a digest query to expand it into its component statements.

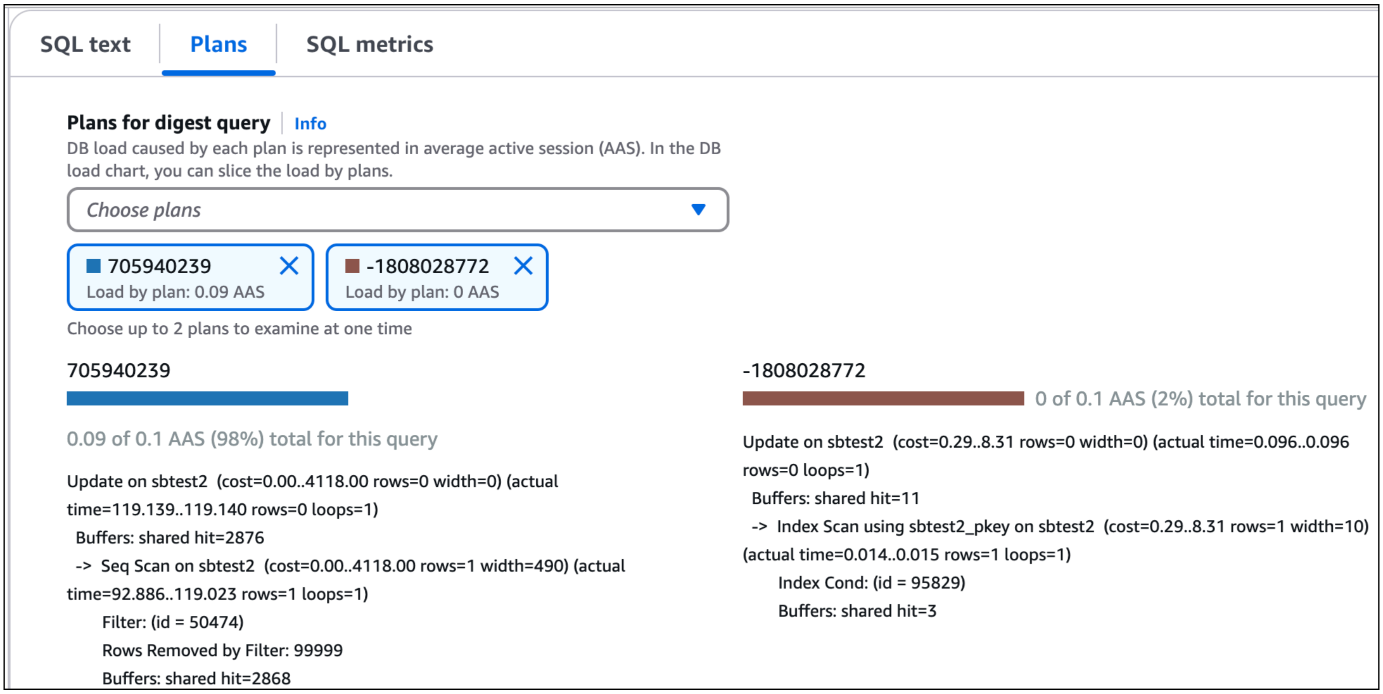


9. Scroll down and view the SQL text. Then, choose the **Plans** tab.

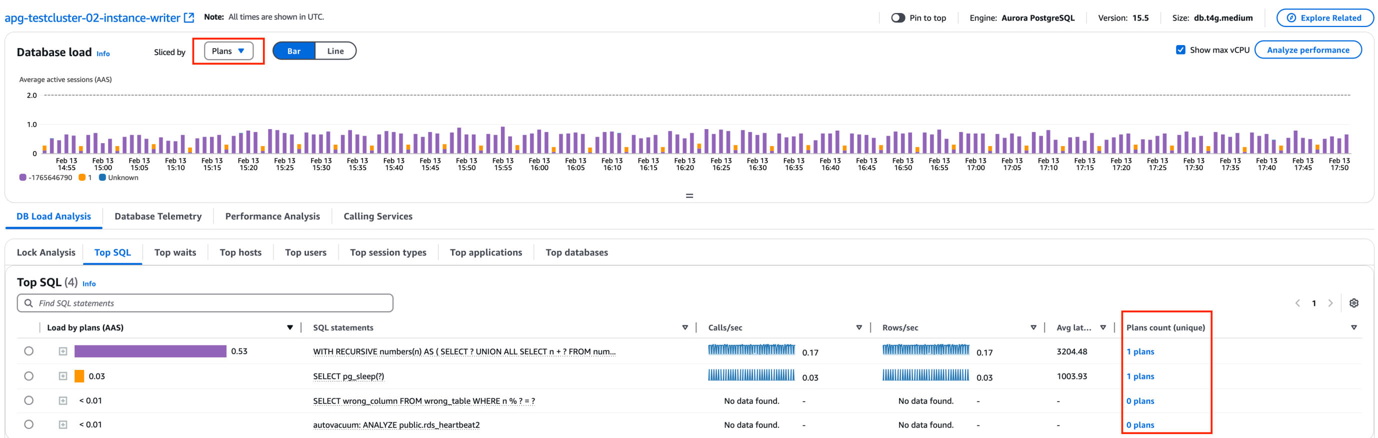
   By default, CloudWatch displays the estimated execution plan. For Aurora PostgreSQL, to view actual execution plans, enable the `aurora_stat_plans.with_analyze` parameter for your DB instance. For more information about the parameter `aurora_stat_plans.with_analyze`, see [Monitoring query execution plans and peak memory for Aurora PostgreSQL](#) in the *Amazon Aurora User Guide*.

10. To compare plans from the same digest query, choose two **Plans** from the **Plans for digest query** list.

    You can view either one or two plans for a query at a time. In the following example screenshot, both plans are for Aurora PostgreSQL.



11. You can also view how each plan contributes to DBLoad over time by choosing **Plans** in the **Slice by** drop-down in the DBLoad chart.



Troubleshooting for CloudWatch Database Insights

Use the following information to troubleshoot issues for CloudWatch Database Insights.

Applying tags to Amazon RDS resources

To apply tags to your databases, use the Amazon RDS API, Amazon CLI, or Amazon RDS console. For more information, see the following topics.

- AddTagsToResource in the *Amazon RDS API Reference*
- add-tags-to-resource in the *Amazon RDS Command Line Reference*
- Tagging Amazon Aurora and Amazon RDS resources in the *Amazon Aurora User Guide*

Maximum DB instances for fleets

You can't monitor more than 500 DB instances in a database fleet. You can use filters to create a fleet health view with less than 500 DB instances.

Use Contributor Insights to analyze high-cardinality data

You can use Contributor Insights to analyze log data and create time series that display contributor data. You can see metrics about the top-N contributors, the total number of unique contributors, and their usage. This helps you find top talkers and understand who or what is impacting system performance. For example, you can find bad hosts, identify the heaviest network users, or find the URLs that generate the most errors.

You can build your rules from scratch, and when you use the Amazon Web Services Management Console you can also use sample rules that Amazon has created. Rules define the log fields that you want to use to define contributors, such as `IpAddress`. You can also filter the log data to find and analyze the behavior of individual contributors.

CloudWatch also provides built-in rules that you can use to analyze metrics from other Amazon services.

All rules analyze incoming data in real time.

If you are signed in to an account that is set up as a monitoring account in CloudWatch cross-account observability, you can create Contributor Insights rules in that monitoring account that analyze log groups in source accounts and in the monitoring account. You can also create a single rule that analyzes log groups in multiple accounts. For more information, see [CloudWatch cross-account observability](#).

> **Important**
>
> If you are using a log transformer on the log group, then when you create a Contributor Insights rule, you must specify the original field key of the logs instead of the transformed field key.

With Contributor Insights, you are charged for each occurrence of a log event that matches a rule. For more information, see [Amazon CloudWatch Pricing](#).

> **Note**
>
> Contributor Insights can only match log entries when the numeric values that the rule references are between -1e9 and 1e9. If a value in a log entry is outside of this range, Contributor Insights skips that log entry.

Topics

Create a Contributor Insights rule in CloudWatch

You can create rules to analyze log data. Any logs in JSON or Common Log Format (CLF) can be evaluated. This includes your custom logs that follow one of these formats and logs from Amazon services such as Amazon VPC flow logs, Amazon Route 53 DNS query logs, Amazon ECS container logs, and logs from Amazon CloudTrail, Amazon SageMaker AI, Amazon RDS, Amazon AppSync and API Gateway.

In a rule, when you specify field names or values, all matching is case sensitive.

You can use built-in sample rules when you create a rule or you can create your own rule from scratch. Contributor Insights includes sample rules for the following types of logs:

- Amazon API Gateway logs
- Amazon Route 53 public DNS query logs
- Amazon Route 53 resolver query logs
- CloudWatch Container Insights logs
- VPC flow logs

If you are signed in to an account that is set up as a monitoring account in CloudWatch cross-account observability, you can create Contributor Insights rules for log groups in the source accounts linked to this monitoring account, in addition to creating rules for log groups in the monitoring account. You can also set up a single rule that monitors log groups in different accounts. For more information, see CloudWatch cross-account observability.

> **Important**
>
> When you grant a user the `cloudwatch:PutInsightRule` permission, by default that user can create a rule that evaluates any log group in CloudWatch Logs. You can add IAM policy conditions that limit these permissions for a user to include and exclude specific log groups. For more information, see Using condition keys to limit Contributor Insights users' access to log groups.

To create a rule using a built-in sample rule

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Logs**, **Contributor Insights**.
3. Choose **Create rule**.
4. For **Select log group(s)**, select the log group(s) that you want your rule to monitor. You can select as many as 20 log groups. If you are signed in to a monitoring account that is set up for CloudWatch cross-account observability, you can select log groups in source accounts, and you can also set up a single rule to analyze log groups in different accounts.

   - (Optional) To select all log groups that have names beginning with a specific string, choose the **Select by prefix match** dropdown, and then enter the prefix. If this is a monitoring account, you can optionally select the accounts to search in, otherwise all accounts are selected.

   > **Note**
   >
   > You incur charges for each log event that matches your rule. If you choose the **Select by prefix match** dropdown, be aware of how many log groups the prefix can match. If you search more log groups than you want, you might incur unexpected charges. For more information, see Amazon CloudWatch Pricing.

5. For **Rule type**, choose **Sample rule**. Then choose **Select sample rule** and select the rule.
6. The sample rule has filled out the **Log format**, **Contribution**, **Filters**, and **Aggregate on** fields. You can adjust those values, if you like.
7. Choose **Next**.

8. For **Rule name**, enter a name. Valid characters are A-Z, a-z, 0-9, (hyphen), (underscore), and (period).
9. Choose whether to create the rule in a disabled or enabled state. If you choose to enable it, the rule immediately starts analyzing your data. You incur costs when you run enabled rules. For more information, see [Amazon CloudWatch Pricing](#).

   Contributor Insights analyzes only new log events after a rule is created. A rule cannot process logs events that were previously processed by CloudWatch Logs.
10. (Optional) For **Tags**, add one or more key-value pairs as tags for this rule. Tags can help you identify and organize your Amazon resources and track your Amazon costs. For more information, see [Tagging your Amazon CloudWatch resources](#).
11. Choose **Create**.

To create a rule from scratch

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Logs**, **Contributor Insights**.
3. Choose **Create rule**.
4. For **Select log group(s)**, select the log group(s) that you want your rule to monitor. You can select as many as 20 log groups. If you are signed in to a monitoring account that is set up for CloudWatch cross-account observability, you can select log groups in source accounts, and you can also set up a single rule to analyze log groups in different accounts.

   - (Optional) To select all log groups that have names beginning with a specific string, choose the **Select by prefix match** dropdown, and then enter the prefix.

   > **Note**
   >
   > You incur charges for each log event that matches your rule. If you choose the **Select by prefix match** dropdown, be aware of how many log groups the prefix can match. If you search more log groups than you want, you might incur unexpected charges. For more information, see [Amazon CloudWatch Pricing](#).
5. For **Rule type**, choose **Custom rule**.
6. For **Log format**, choose **JSON** or **CLF**.

7. You can finish creating the rule by using the wizard or by choosing the **Syntax** tab and specifying your rule syntax manually.

   To continue using the wizard, do the following:

   a. For **Contribution**, **Key**, enter a contributor type that you want to report on. The report displays the top-N values for this contributor type.

      Valid entries are any log field that has values. Examples include **`requestId`**, **`sourceIPaddress`**, and **`containerID`**.

      For information about finding the log field names for the logs in a certain log group, see [Finding Log Fields](#).

      Keys larger than 1 KB are truncated to 1KB.

   b. (Optional) Choose **Add new key** to add more keys. You can include as many as four keys in a rule. If you enter more than one key, the contributors in the report are defined by unique value combinations of the keys. For example, if you specify three keys, each unique combination of values for the three keys is counted as a unique contributor.

   c. (Optional) If you want to add a filter that narrows the scope of your results, choose **Add filter**. For **Match**, enter the name of the log field that you want to filter on. For **Condition**, choose a comparison operator, and enter a value that you want to filter for.

      You can add as many as four filters in a rule. Multiple filters are joined by AND logic, so only log events that match all filters are evaluated.

      > **Note**
      >
      > Arrays that follow comparison operators, such as `In`, `NotIn`, or `StartsWith`, can include as many as 10 string values. For more information about the Contributor Insights rules syntax, see [Contributor Insights rule syntax in CloudWatch](#).

   d. For **Aggregate on**, choose **Count** or **Sum**. Choosing **Count** causes the contributor ranking to be based on the number of occurrences. Choosing **Sum** causes the ranking to be based on the aggregated sum of the values of the field that you specify for **Contribution**, **Value**.

8. To enter your rule as a JSON object instead of using the wizard, do the following:

   a. Choose the **Syntax** tab.

   b. In **Rule body**, enter the JSON object for your rule. For information about rule syntax, see [Contributor Insights rule syntax in CloudWatch](#).

9. Choose **Next**.
10. For **Rule name**, enter a name. Valid characters are A-Z, a-z, 0-9, "-", "_", and ".".
11. Choose whether to create the rule in a disabled or enabled state. If you choose to enable it, the rule immediately starts analyzing your data. You incur costs when you run enabled rules. For more information, see [Amazon CloudWatch Pricing](#).

    Contributor Insights analyzes only new log events after a rule is created. A rule cannot process logs events that were previously processed by CloudWatch Logs.
12. (Optional) For **Tags**, add one or more key-value pairs as tags for this rule. Tags can help you identify and organize your Amazon resources and track your Amazon costs. For more information, see [Tagging your Amazon CloudWatch resources](#).
13. Choose **Next**.
14. Confirm the settings that you entered, and choose **Create rule**.

You can disable, enable, or delete rules that you have created.

To enable, disable, or delete a rule in Contributor Insights

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Logs**, **Contributor Insights**.
3. In the list of rules, select the check box next to a single rule.

   Built-in rules are created by Amazon services and can't be edited, disabled, or deleted.
4. Choose **Actions**, and then choose the option you want.

Finding log fields

When you create a rule, you need to know the names of fields in the log entries in a log group.

To find the log fields in a log group

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, under **Logs**, choose **Insights**.

3. Above the query editor, select one or more log groups to query.

   When you select a log group, CloudWatch Logs Insights automatically detects fields in the data in the log group and displays them in the right pane in **Discovered fields**.

Contributor Insights rule syntax in CloudWatch

This section explains the syntax for Contributor Insights rules. Use this syntax only when you are creating a rule by entering a JSON block. If you use the wizard to create a rule, you don't need to know the syntax. For more information about creating rules using the wizard, see [Create a Contributor Insights rule in CloudWatch](#).

All matching of rules to log event field names and values is case sensitive.

The following example illustrates the syntax for JSON logs.

```
{
    "Schema": {
        "Name": "CloudWatchLogRule",
        "Version": 1
    },
    "LogGroupNames": [
        "API-Gateway-Access-Logs*",
        "Log-group-name2"
    ],
    "LogFormat": "JSON",
    "Contribution": {
        "Keys": [
            "$.ip"
        ],
        "ValueOf": "$.requestBytes",
        "Filters": [
            {
                "Match": "$.httpMethod",
                "In": [
                    "PUT"
                ]
            }
        ]
    },
    "AggregateOn": "Sum"
}
```

Fields in Contributor Insights rules

Schema

The value of `Schema` for a rule that analyzes CloudWatch Logs data must always be `{"Name": "CloudWatchLogRule", "Version": 1}`

LogGroupNames

An array of strings. For each element in the array, you can optionally use `*` at the end of a string to include all log groups with names that start with that prefix.

Be careful about using wildcards with log group names. You incur charges for each log event that matches a rule. If you accidentally search more log groups than you intend, you might incur unexpected charges. For more information, see Amazon CloudWatch Pricing.

LogGroupARNs

If you are creating this rule in a CloudWatch cross-account observability monitoring account, you can use `LogGroupARNs` to specify log groups in source accounts that are linked to the monitoring account, and to specify log groups in the monitoring account itself. You must specify either `LogGroupNames` or `LogGroupARNs` in your rule, but not both.

`LogGroupARNs` is an array of strings. For each element in the array, you can optionally use `*` as a wildcard in certain situations. For example you can specify `arn:aws:logs:us-west-1:*:log-group/MyLogGroupName2` to specify log groups named `MyLogGroupName2` in all source accounts and in the monitoring account, in the US West (N. California) Region. You can also specify `arn:aws:logs:us-west-1:111122223333:log-group/GroupNamePrefix*` to specify all log groups in US West (N. California) in 111122223333 that have names starting with `GroupNamePrefix`.

You can't specify a partial Amazon account ID as a prefix with a wild card.

Be careful about using wildcards with log group ARNs. You incur charges for each log event that matches a rule. If you accidentally search more log groups than you intend, you might incur unexpected charges. For more information, see Amazon CloudWatch Pricing.

LogFormat

Valid values are `JSON` and `CLF`.

Contribution

This object includes a `Keys` array with as many as four members, optionally a single `ValueOf`, and optionally an array of as many as four `Filters`.

Keys

An array of up to four log fields that are used as dimensions to classify contributors. If you enter more than one key, each unique combination of values for the keys is counted as a unique contributor. The fields must be specified using JSON property format notation.

ValueOf

(Optional) Specify this only when you are specifying `Sum` as the value of `AggregateOn`. `ValueOf` specifies a log field with numerical values. In this type of rule, the contributors are ranked by their sum of the value of this field, instead of their number of occurrences in the log entries. For example, if you want to sort contributors by their total `BytesSent` over a period, you would set `ValueOf` to `BytesSent` and specify `Sum` for `AggregateOn`.

Filters

Specifies an array of as many as four filters to narrow the log events that are included in the report. If you specify multiple filters, Contributor Insights evaluates them with a logical AND operator. You can use this to filter out irrelevant log events in your search or you can use it to select a single contributor to analyze their behavior.

Each member in the array must include a `Match` field and a field indicating the type of matching operator to use.

The `Match` field specifies a log field to evaluate in the filter. The log field is specified using JSON property format notation.

The matching operator field must be one of the following: `In`, `NotIn`, `StartsWith`, `GreaterThan`, `LessThan`, `EqualTo`, `NotEqualTo`, or `IsPresent`. If the operator field is `In`, `NotIn`, or `StartsWith`, it is followed by an array of string values to check for. Contributor Insights evaluates the array of string values with an OR operator. The array can include as many as 10 string values.

If the operator field is `GreaterThan`, `LessThan`, `EqualTo`, or `NotEqualTo`, it is followed by a single numerical value to compare with.

If the operator field is `IsPresent`, it is followed by either `true` or `false`. This operator matches log events based on whether the specified log field is present in the log event. The `isPresent` works only with values in the leaf node of JSON properties. For example, a filter that looks for matches to `c-count` does not evaluate a log event with a value of `details.c-count.c1`.

See the following four filter examples:

```
{"Match": "$.httpMethod", "In": [ "PUT", ] }
{"Match": "$.StatusCode", "EqualTo": 200 }
{"Match": "$.BytesReceived", "GreaterThan": 10000}
{"Match": "$.eventSource", "StartsWith": [ "ec2", "ecs" ] }
```

AggregateOn

Valid values are `Count` and `Sum`. Specifies whether to aggregate the report based on a count of occurrences or a sum of the values of the field that is specified in the `ValueOf` field.

JSON property format notation

The `Keys`, `ValueOf`, and `Match` fields follow JSON property format with dot notation, where $ represents the root of the JSON object. This is followed by a period and then an alphanumeric string with the name of the subproperty. Multiple property levels are supported.

The first character of the string can only be A-Z or a-z. The following characters of the string can be A-Z, a-z, or 0-9.

The following list illustrates valid examples of JSON property format:

```
$.userAgent
$.endpoints[0]
$.users[1].name
$.requestParameters.instanceId
```

Additional field in rules for CLF logs

Common Log Format (CLF) log events do not have names for the fields like JSON does. To provide the fields to use for Contributor Insights rules, a CLF log event can be treated as array with an index starting from 1. You can specify the first field as **"1"**, the second field as **"2"**, and so on.

To make a rule for a CLF log easier to read, you can use `Fields`. This enables you to provide a naming alias for CLF field locations. For example, you can specify that the location "4" is an IP address. Once specified, `IpAddress` can be used as property in the `Keys`, `ValueOf`, and `Filters` in the rule.

The following is an example of a rule for a CLF log that uses the `Fields` field.

```
{
    "Schema": {
        "Name": "CloudWatchLogRule",
        "Version": 1
    },
    "LogGroupNames": [
        "API-Gateway-Access-Logs*"
    ],
    "LogFormat": "CLF",
    "Fields": {
        "4": "IpAddress",
        "7": "StatusCode"
    },
    "Contribution": {
        "Keys": [
            "IpAddress"
        ],
        "Filters": [
            {
                "Match": "StatusCode",
                "EqualTo": 200
            }
        ]
    },
    "AggregateOn": "Count"
}
```

CloudWatch Contributor Insights rule examples

This section contains examples that illustrate use cases for Contributor Insights rules.

VPC Flow Logs: Byte transfers by source and destination IP address

```
{
    "Schema": {
        "Name": "CloudWatchLogRule",
        "Version": 1
    },
```

```
    "LogGroupNames": [
        "/aws/containerinsights/sample-cluster-name/flowlogs"
    ],
    "LogFormat": "CLF",
    "Fields": {
        "4": "srcaddr",
        "5": "dstaddr",
        "10": "bytes"
    },
    "Contribution": {
        "Keys": [
            "srcaddr",
            "dstaddr"
        ],
        "ValueOf": "bytes",
        "Filters": []
    },
    "AggregateOn": "Sum"
}
```

VPC Flow Logs: Highest number of HTTPS requests

```
{
    "Schema": {
        "Name": "CloudWatchLogRule",
        "Version": 1
    },
    "LogGroupNames": [
        "/aws/containerinsights/sample-cluster-name/flowlogs"
    ],
    "LogFormat": "CLF",
    "Fields": {
        "5": "destination address",
        "7": "destination port",
        "9": "packet count"
    },
    "Contribution": {
        "Keys": [
            "destination address"
        ],
        "ValueOf": "packet count",
        "Filters": [
            {
```

```
                "Match": "destination port",
                "EqualTo": 443
            }
        ]
    },
    "AggregateOn": "Sum"
}
```

VPC Flow Logs: Rejected TCP connections

```
{
    "Schema": {
        "Name": "CloudWatchLogRule",
        "Version": 1
    },
    "LogGroupNames": [
        "/aws/containerinsights/sample-cluster-name/flowlogs"
    ],
    "LogFormat": "CLF",
    "Fields": {
        "3": "interfaceID",
        "4": "sourceAddress",
        "8": "protocol",
        "13": "action"
    },
    "Contribution": {
        "Keys": [
            "interfaceID",
            "sourceAddress"
        ],
        "Filters": [
            {
                "Match": "protocol",
                "EqualTo": 6
            },
            {
                "Match": "action",
                "In": [
                    "REJECT"
                ]
            }
        ]
    },
```

```
    "AggregateOn": "Sum"
}
```

Route 53 NXDomain responses by source address

```
{
    "Schema": {
        "Name": "CloudWatchLogRule",
        "Version": 1
    },
    "AggregateOn": "Count",
    "Contribution": {
        "Filters": [
            {
                "Match": "$.rcode",
                "StartsWith": [
                    "NXDOMAIN"
                ]
            }
        ],
        "Keys": [
            "$.srcaddr"
        ]
    },
    "LogFormat": "JSON",
    "LogGroupNames": [
        "<loggroupname>"
    ]
}
```

Route 53 resolver queries by domain name

```
{
    "Schema": {
        "Name": "CloudWatchLogRule",
        "Version": 1
    },
    "AggregateOn": "Count",
    "Contribution": {
        "Filters": [],
        "Keys": [
            "$.query_name"
        ]
```

```
    },
    "LogFormat": "JSON",
    "LogGroupNames": [
        "<loggroupname>"
    ]
}
```

Route 53 resolver queries by query type and source address

```
{
    "Schema": {
        "Name": "CloudWatchLogRule",
        "Version": 1
    },
    "AggregateOn": "Count",
    "Contribution": {
        "Filters": [],
        "Keys": [
            "$.query_type",
            "$.srcaddr"
        ]
    },
    "LogFormat": "JSON",
    "LogGroupNames": [
        "<loggroupname>"
    ]
}
```

Viewing Contributor Insights reports in CloudWatch

To view graphs of report data and a ranked list of contributors found by your rules, follow these steps.

To view your rule reports

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Logs**, **Contributor Insights**.
3. In the list of rules, choose the name of a rule.

   The graph displays the results of the rule over the last three hours. The table under the graph shows the top 10 contributors.

4. To change the number of contributors shown in the table, choose **Top 10 contributors** at the top of the graph.
5. To filter the graph to show only the results from a single contributor, choose that contributor in the table legend. To again show all contributors, choose that same contributor again in the legend.
6. To change the time range shown in the report, choose **15m**, **30m**, **1h**, **2h**, **3h**, or **custom** at the top of the graph.

   The maximum time range for the report is 24 hours, but you can choose a 24-hour window that occurred up to 15 days ago. To choose a time window in the past, choose **custom**, **absolute**, and then specify your time window.
7. To change the length of the time period used for the aggregation and ranking of contributors, choose **period** at the top of the graph. Viewing a longer time period generally shows a smoother report with few spikes. Choosing a shorter time period is more likely to display spikes.
8. To add this graph to a CloudWatch dashboard, choose **Add to dashboard**.
9. To open the CloudWatch Logs Insights query window, with the log groups in this report already loaded in the query box, choose **View logs**.
10. To export the report data to your clipboard or a CSV file, choose **Export**.

Graphing metrics generated by rules in CloudWatch

Contributor Insights provides a metric math function, `INSIGHT_RULE_METRIC`. You can use this function to add data from a Contributor Insights report to a graph in the **Metrics** tab of the CloudWatch console. You can also set an alarm based on this math function. For more information about metric math functions, see [Using math expressions with CloudWatch metrics](#).

To use this metric math function, you must be signed in to an account that has both the `cloudwatch:GetMetricData` and `cloudwatch:GetInsightRuleReport` permissions.

The syntax is `INSIGHT_RULE_METRIC(ruleName, metricName)`. *`ruleName`* is the name of a Contributor Insights rule. *`metricName`* is one of the values in the following list. The value of *`metricName`* determines which type of data the math function returns.

- `UniqueContributors` — the number of unique contributors for each data point.

- MaxContributorValue — the value of the top contributor for each data point. The identity of the contributor might change for each data point in the graph.

  If this rule aggregates by Count, the top contributor for each data point is the contributor with the most occurrences in that period. If the rule aggregates by Sum, the top contributor is the contributor with the greatest sum in the log field specified by the rule's Value during that period.
- SampleCount — the number of data points matched by the rule.
- Sum — the sum of the values from all contributors during the time period represented by that data point.
- Minimum — the minimum value from a single observation during the time period represented by that data point.
- Maximum — the maximum value from a single observation during the time period represented by that data point.
- Average — the average value from all contributors during the time period represented by that data point.

Setting an alarm on Contributor Insights metric data

Using the function INSIGHT_RULE_METRIC, you can set alarms on metrics that Contributor Insights generates. For example, you can create an alarm that's based on the percentage of rejected transmission control protocol (TCP) connections. To get started with this type of alarm, you can create rules like the ones shown in the following two examples:

Example rule: "RejectedConnectionsRule"

```
{
    "Schema": {
        "Name": "CloudWatchLogRule",
        "Version": 1
    },
    "LogGroupNames": [
        "/aws/containerinsights/sample-cluster-name/flowlogs"
    ],
    "LogFormat": "CLF",
    "Fields": {
        "3": "interfaceID",
        "4": "sourceAddress",
```

```
        "8": "protocol",
        "13": "action"
    },
    "Contribution": {
        "Keys": [
            "interfaceID",
            "sourceAddress"
        ],
        "Filters": [
            {
                "Match": "protocol",
                "EqualTo": 6
            },
            {
                "Match": "action",
                "In": [
                    "REJECT"
                ]
            }
        ]
    },
    "AggregateOn": "Sum"
}
```

Example rule: "TotalConnectionsRule"

```
{
    "Schema": {
        "Name": "CloudWatchLogRule",
        "Version": 1
    },
    "LogGroupNames": [
        "/aws/containerinsights/sample-cluster-name/flowlogs"
    ],
    "LogFormat": "CLF",
    "Fields": {
        "3": "interfaceID",
        "4": "sourceAddress",
        "8": "protocol",
        "13": "action"
    },
    "Contribution": {
        "Keys": [
```

```
            "interfaceID",
            "sourceAddress"
        ],
        "Filters": [{
            "Match": "protocol",
            "EqualTo": 6
        }],
        "AggregateOn": "Sum"
    }
}
```

After you create your rules, you can select the **Metrics** tab in the CloudWatch Console, where you can use the following example metric math expressions to graph the data that Contributor Insights reports:

Example: Metric math expressions

```
e1 INSIGHT_RULE_METRIC("RejectedConnectionsRule", "Sum")
e2 INSIGHT_RULE_METRIC("TotalConnectionsRule", "Sum")
e3 (e1/e2)*100
```

In the example, the metric math expression `e3` returns all of the rejected TCP connections. If you want to be notified when 20 percent of the TCP connections are rejected, you can modify the expression by changing the threshold from `100` to `20`.

> **Note**
>
> You can set an alarm on a metric that you're monitoring from the **Metrics** section. While on the **Graphed metrics** tab, you can select the **Create alarm** icon under the **Actions** column. The **Create alarm** icon looks like a bell.

For more information about graphing metrics and using metric math functions, see the following section: [Add a math expression to a CloudWatch graph](#).

Using Contributor Insights built-in rules in CloudWatch

You can use Contributor Insights built-in rules to analyze metrics from other Amazon services. The following services support built-in rules:

- [Contributor Insights for Amazon DynamoDB](#) in the *Amazon DynamoDB Developer Guide*.

- Use built-in Contributor Insights rules in the *Amazon PrivateLink Guide*.

Detect common application problems with CloudWatch Application Insights

You can use Amazon CloudWatch Application Insights to detect problems with your applications. CloudWatch Application Insights facilitates observability for your applications and underlying Amazon resources. It helps you set up the best monitors for your application resources to continuously analyze data for signs of problems with your applications. Application Insights, which is powered by SageMaker and other Amazon technologies, provides automated dashboards that show potential problems with monitored applications, which help you to quickly isolate ongoing issues with your applications and infrastructure. The enhanced visibility into the health of your applications that Application Insights provides helps reduce mean time to repair (MTTR) to troubleshoot your application issues.

When you add your applications to Amazon CloudWatch Application Insights, it scans the resources in the applications and recommends and configures metrics and logs on CloudWatch for application components. Example application components include SQL Server backend databases and Microsoft IIS/Web tiers. Application Insights analyzes metric patterns using historical data to detect anomalies, and continuously detects errors and exceptions from your application, operating system, and infrastructure logs. It correlates these observations using a combination of classification algorithms and built-in rules. Then, it automatically creates dashboards that show the relevant observations and problem severity information to help you prioritize your actions. For common problems in .NET and SQL application stacks, such as application latency, SQL Server failed backups, memory leaks, large HTTP requests, and canceled I/O operations, it provides additional insights that point to a possible root cause and steps for resolution. Built-in integration with Amazon SSM OpsCenter allows you to resolve issues by running the relevant Systems Manager Automation document.

Sections

- Work with component configurations
- Create and configure CloudWatch Application Insights monitoring using CloudFormation templates
- Tutorial: Set up monitoring for SAP ASE
- Tutorial: Set up monitoring for SAP HANA
- Tutorial: Set up monitoring for SAP NetWeaver
- View and troubleshoot problems detected by Amazon CloudWatch Application Insights
- Logs and metrics supported by Amazon CloudWatch Application Insights

What is Amazon CloudWatch Application Insights?

CloudWatch Application Insights helps you monitor your applications that use Amazon EC2 instances along with other application resources. It identifies and sets up key metrics, logs, and alarms across your application resources and technology stack (for example, your Microsoft SQL Server database, web (IIS) and application servers, OS, load balancers, and queues). It continuously monitors metrics and logs to detect and correlate anomalies and errors. When errors and anomalies are detected, Application Insights generates CloudWatch Events that you can use to set up notifications or take actions. To assist with troubleshooting, it creates automated dashboards for detected problems, which include correlated metric anomalies and log errors, along with additional insights to point you to a potential root cause. The automated dashboards help you to take remedial actions to keep your applications healthy and to prevent impact to the end-users of your application. It also creates OpsItems so that you can resolve problems using Amazon SSM OpsCenter.

You can configure important counters, such as Mirrored Write Transaction/sec, Recovery Queue Length, and Transaction Delay, as well as Windows Event Logs on CloudWatch. When a failover event or problem occurs with your SQL HA workload, such as a restricted access to query a target database, CloudWatch Application Insights provides automated insights .

CloudWatch Application Insights integrates with Amazon Launch Wizard to provide a one-click monitoring setup experience for deploying SQL Server HA workloads on Amazon. When you select the option to set up monitoring and insights with Application Insights on the Launch Wizard console, CloudWatch Application Insights automatically sets up relevant metrics, logs, and alarms on CloudWatch, and starts monitoring newly deployed workloads. You can view automated insights and detected problems, along with the health of your SQL Server HA workloads, on the CloudWatch console.

Contents

Features

Application Insights provides the following features.

Automatic set up of monitors for application resources

CloudWatch Application Insights reduces the time it takes to set up monitoring for your applications. It does this by scanning your application resources, providing a customizable list of recommended metrics and logs, and setting them up on CloudWatch to provide necessary visibility into your application resources, such as Amazon EC2 and Elastic Load Balancers (ELB). It also sets up dynamic alarms on monitored metrics. The alarms are automatically updated based on anomalies detected in the previous two weeks.

Problem detection and notification

CloudWatch Application Insights detects signs of potential problems with your application, such as metric anomalies and log errors. It correlates these observations to surface potential problems with your application. It then generates CloudWatch Events, which can be configured to receive notifications or take actions. This eliminates the need for you to create individual alarms on metrics or log errors. Additionally, you can Configure Amazon SNS notifications to receive alerts for detected problems.

Troubleshooting

CloudWatch Application Insights creates CloudWatch automatic dashboards for problems that are detected. The dashboards show details about the problem, including the associated metric anomalies and log errors to help you with troubleshooting. They also provide additional insights that point to potential root causes of the anomalies and errors.

Concepts

The following concepts are important for understanding how Application Insights monitors your application.

Component

An auto-grouped, standalone, or custom grouping of similar resources that make up an application. We recommend grouping similar resources into custom components for better monitoring.

Observation

An individual event (metric anomaly, log error, or exception) that is detected with an application or application resource.

Problem

Problems are detected by correlating, classifying, and grouping related observations.

For definitions of other key concepts for CloudWatch Application Insights, see [Amazon CloudWatch Concepts](#).

Pricing

CloudWatch Application Insights sets up recommended metrics and logs for selected application resources using CloudWatch metrics, Logs, and Events for notifications on detected problems. These features are charged to your Amazon account according to CloudWatch pricing. For detected problems, SSM OpsItems are also created by Application Insights to notify you about problems. Additionally, Application Insights creates SSM Parameter Store parameters to configure the CloudWatch agents on your instances. The Amazon EC2 Systems Manager features are charged according to SSM pricing. You are not charged for setup assistance, monitoring, data analysis, or problem detection.

Costs for CloudWatch Application Insights

Costs for Amazon EC2 include usage of the following features:

- CloudWatch Agent
  - CloudWatch Agent log groups
  - CloudWatch Agent metrics

- Prometheus log groups (for JMX workloads)

Costs for all resources include usage of the following features:

- CloudWatch alarms (majority of cost)
- SSM OpsItems (minimal cost)

Example cost calculation

The costs in this example are considered according to the following scenario.

You created a resource group that includes the following:

- An Amazon EC2 instance with SQL Server installed.
- An attached Amazon EBS volume.

When you onboard this resource group with CloudWatch Application Insights, the SQL Server workload installed on the Amazon EC2 instance is detected. CloudWatch Application Insights starts monitoring the following metrics.

The following metrics are monitored for the SQL Server instance:

- CPUUtilization
- StatusCheckFailed
- Memory % Committed Bytes in Use
- Memory Available Mbytes
- Network Interface Bytes Total/sec
- Paging File % Usage
- Physical Disk % Disk Time
- Processor % Processor Time
- SQLServer:Buffer Manager cache hit ratio
- SQLServer:Buffer Manager life expectancy
- SQLServer:General Statistics Processes blocked
- SQLServer:General Statistics User Connections
- SQLServer:Locks Number of Deadlocks/sec

- SQLServer:SQL Statistics Batch Requests/sec
- System Processor Queue Length

The following metrics are monitored for the volumes attached to the SQL Server instance:

- VolumeReadBytes
- VolumeWriteBytes
- VolumeReadOps
- VolumeWriteOps
- VolumeTotalReadTime
- VolumeTotalWriteTime
- VolumeIdleTime
- VolumeQueueLength
- VolumeThroughputPercentage
- VolumeConsumedReadWriteOps
- BurstBalance

For this scenario, the costs are calculated according to the CloudWatch pricing page and the SSM pricing page:

- **Custom metrics**

  For this scenario, 13 of the above metrics are emitted to CloudWatch using the CloudWatch agent. These metrics are treated as custom metrics. The cost for each custom metric is $.3/month. The total cost for these custom metrics is 13 * $.3 = $3.90/month.
- **Alarms**

  For this scenario, CloudWatch Application Insights monitors 26 metrics in total, which creates 26 alarms. The cost for each alarm is $.1/month. The total cost for alarms is 26 * $.1 = $2.60/month.
- **Data ingestion and error logs**

  The cost of data ingestion is $.05/GB and storage for the SQL Server error log is $.03/GB. The total cost for data ingestion and the error log is $.05/GB + $.03/GB= $.08/GB.
- **Amazon EC2 Systems Manager OpsItems**

An SSM OpsItem is created for each problem detected by CloudWatch Application Insights. For *n* number of problems in your application, the total cost is $.00267 * *n*/month.

Related services

The following services are used along with CloudWatch Application Insights:

Related Amazon services

- **Amazon CloudWatch** provides system-wide visibility into resource utilization, application performance, and operational health. It collects and tracks metrics, sends alarm notifications, automatically updates resources that you are monitoring based on the rules that you define, and allows you to monitor your own custom metrics. CloudWatch Application Insights is initiated through CloudWatch—specifically, within the CloudWatch default operational dashboards. For more information, see the *Amazon CloudWatch User Guide*.
- **CloudWatch Container Insights** collects, aggregates, and summarizes metrics and logs from your containerized applications and microservices. You can use Container Insights to monitor Amazon ECS, Amazon Elastic Kubernetes Service, and Kubernetes platforms on Amazon EC2. When Application Insights is enabled on the Container Insights or Application Insights consoles, Application Insights displays detected problems on your Container Insights dashboard. For more information, see Container Insights .
- **Amazon DynamoDB** is a fully managed NoSQL database service that lets you offload the administrative burdens of operating and scaling a distributed database so that you don't have to worry about hardware provisioning, setup and configuration, replication, software patching, or cluster scaling. DynamoDB also offers encryption at rest, which eliminates the operational burden and complexity involved in protecting sensitive data.
- **Amazon EC2** provides scalable computing capacity in the Amazon Cloud. You can use Amazon EC2 to launch as many or as few virtual servers as you need, to configure security and networking, and to manage storage. You can scale up or down to handle changes in requirements or spikes in popularity, which reduces your need to forecast traffic. For more information, see the *Amazon EC2 User Guide for Linux Instances* or *Amazon EC2 Guide for Windows Instances*.
- **Amazon Elastic Block Store (Amazon EBS)** provides block-level storage volumes for use with Amazon EC2 instances. Amazon EBS volumes behave like raw, unformatted block devices. You can mount these volumes as devices on your instances. Amazon EBS volumes that are attached to an instance are exposed as storage volumes that persist independently from the life of the

instance. You can create a file system on top of these volumes, or use them in any way you would use a block device (such as a hard drive). You can dynamically change the configuration of a volume attached to an instance. For more information, see the Amazon EBS User Guide.

- **Amazon EC2 Auto Scaling** helps ensure that you have the correct number of EC2 instances available to handle the load for your application. For more information, see the *Amazon EC2 Auto Scaling User Guide*.
- **Elastic Load Balancing** distributes incoming applications or network traffic across multiple targets, such as EC2 instances, containers, and IP addresses, in multiple Availability Zones. For more information, see the *Elastic Load Balancing User Guide*.
- **IAM** is a web service that helps you to securely control access to Amazon resources for your users. Use IAM to control who can use your Amazon resources (authentication), and to control the resources they can use and how they can use them (authorization). For more information, see Authentication and Access Control for Amazon CloudWatch.
- **Amazon Lambda** lets you build serverless applications composed of functions that are triggered by events and automatically deploy them using CodePipeline and Amazon CodeBuild. For more information, see Amazon Lambda Applications.
- **Amazon Launch Wizard for SQL Server** reduces the time it takes to deploy SQL Server high availability solution to the cloud. You input your application requirements, including performance, number of nodes, and connectivity on the service console, and Amazon Launch Wizard identifies the right Amazon resources to deploy and run your SQL Server Always On application.
- **Amazon Resource Groups** help you to organize the resources that make up your application. With Resource Groups, you can manage and automate tasks on a large number of resources at one time. Only one Resource Group can be registered for a single application. For more information, see the *Amazon Resource Groups User Guide*.
- **Amazon SQS** offers a secure, durable, and available hosted queue that allows you to integrate and decouple distributed software systems and components. For more information, see the *Amazon SQS User Guide*.
- **Amazon Step Functions** is a serverless function composer that allows you to sequence a variety of Amazon services and resources, including Amazon Lambda functions, into structured, visual workflows. For more information, see the Amazon Step Functions User Guide.
- **Amazon SSM OpsCenter** aggregates and standardizes OpsItems across services while providing contextual investigation data about each OpsItem, related OpsItems, and related resources. OpsCenter also provides Systems Manager Automation documents (runbooks) that you can use to quickly resolve issues. You can specify searchable, custom data for each OpsItem. You can also

view automatically-generated summary reports about OpsItems by status and source. For more information, see the *Amazon Systems Manager User Guide*.

- **Amazon API Gateway** is an Amazon service for creating, publishing, maintaining, monitoring, and securing REST, HTTP, and WebSocket APIs at any scale. API developers can create APIs that access Amazon or other web services, as well as data stored in the Amazon Cloud. For more information, see the Amazon API Gateway User Guide.

> **Note**
>
> Application Insights supports only REST API protocols (v1 of the API Gateway service).

- **Amazon Elastic Container Service (Amazon ECS)** is a fully managed container orchestration service. You can use Amazon ECS to run your most sensitive and mission-critical applications. For more information, see the Amazon Elastic Container Service Developer Guide.
- **Amazon Elastic Kubernetes Service (Amazon EKS)** is a managed service that you can use to run Kubernetes on Amazon without having to install, operate, and maintain your own Kubernetes control plane or nodes. Kubernetes is an open-source system for automating the deployment, scaling, and management of containerized applications. For more information, see the Amazon EKS User Guide.
- **Kubernetes on Amazon EC2**. Kubernetes is open-source software that helps you deploy and manage containerized applications at scale. Kubernetes manages clusters of Amazon EC2 compute instances and runs containers on those instances with processes for deployment, maintenance, and scaling. With Kubernetes, you can run any type of containerized application with the same toolset on-premises and in the cloud. For more information, see Kubernetes Documentation: Getting started.
- **Amazon FSx** helps you to launch and run popular file systems that are fully managed by Amazon. With Amazon FSx, you can leverage the feature sets and performance of common open source and commercially-licensed file systems to avoid time-consuming administrative tasks. For more information, see the Amazon FSx Documentation.
- **Amazon Simple Notification Service (SNS)** is a fully-managed messaging service for both application-to-application and application-to-person communication. You can configure Amazon SNS for monitoring by Application Insights. When Amazon SNS is configured as a resource for monitoring, Application Insights tracks SNS metrics to help determine why SNS messages may encounter issues or fail.
- **Amazon Elastic File System (Amazon EFS)** is a fully-managed elastic NFS file system for use with Amazon Web Services Cloud services and on-premises resources. It is built to scale to

petabytes on demand without disrupting applications. It grows and shrinks automatically as you add and remove files, which eliminates the need to provision and manage capacity to accommodate growth. For more information, see the Amazon Elastic File System documentation.

Related third-party services

- For some workloads and applications monitored in Application Insights, **Prometheus JMX exporter** is installed using Amazon Systems Manager Distributor so that CloudWatch Application Insights can retrieve Java-specific metrics. When you choose to monitor a Java application, Application Insights automatically installs the Prometheus JMX exporter for you.

Supported application components

CloudWatch Application Insights scans your resource group to identify application components. Components can be standalone, auto-grouped (such as instances in an Auto Scaling group or behind a load balancer), or custom (by grouping together individual Amazon EC2 instances).

The following components are supported by CloudWatch Application Insights:

Amazon components

- Amazon EC2
- Amazon EBS
- Amazon RDS
- Elastic Load Balancing: Application Load Balancer and Classic Load Balancer (all target instances of these load balancers are identified and configured).
- Amazon EC2 Auto Scaling groups: Amazon Auto Scaling (Auto Scaling groups are dynamically configured for all target instances; if your application scales up, CloudWatch Application Insights automatically configures the new instances). Auto Scaling groups are not supported for CloudFormation stack-based resource groups.
- Amazon Lambda
- Amazon Simple Queue Service (Amazon SQS)
- Amazon DynamoDB table
- Amazon S3 bucket metrics
- Amazon Step Functions

- Amazon API Gateway REST API stages
- Amazon Elastic Container Service (Amazon ECS): cluster, service, and task
- Amazon Elastic Kubernetes Service (Amazon EKS): cluster
- Kubernetes on Amazon EC2: Kubernetes cluster running on EC2
- Amazon SNS topic

Any other component type resources are not currently tracked by CloudWatch Application Insights. If a component type that is supported does not appear in your Application Insights application, the component may already be registered and managed by another application you own that is monitored by Application Insights.

Supported technology stacks

You can use CloudWatch Application Insights to monitor your applications running on Windows Server and Linux operating systems by selecting the application tier dropdown menu option for one of the following technologies:

- Front-end: Microsoft Internet Information Services (IIS) Web Server
- Worker-tier:
  - .NET Framework
  - .NET Core
- Applications:
  - Java
  - SAP NetWeaver standard, distributed, and high availability deployments
- Active Directory
- SharePoint
- Databases:
  - Microsoft SQL Server running on Amazon RDS or Amazon EC2 (including SQL Server High Availability configurations. See, Component configuration examples).
  - MySQL running on Amazon RDS, Amazon Aurora, or Amazon EC2
  - PostgreSQL running on Amazon RDS or Amazon EC2
  - Amazon DynamoDB table
  - Oracle running on Amazon RDS or Amazon EC2

- SAP HANA database on a single Amazon EC2 instance and multiple EC2 instances
- Cross-AZ SAP HANA database high availability setup
- SAP Sybase ASE database on a single Amazon EC2 instance
- Cross-AZ SAP Sybase ASE database high availability setup

If none of the technology stacks listed above apply to your application resources, you can monitor your application stack by choosing **Custom** from the application tier dropdown menu on the **Manage monitoring** page.

How Amazon CloudWatch Application Insights works

CloudWatch Application Insights provides monitoring of your application resources. The following information describes how Application Insights works.

Topics

How Application Insights monitors applications

The following information describes how Application Insights monitors applications.

Application discovery and configuration

The first time an application is added to CloudWatch Application Insights it scans the application components to recommend key metrics, logs, and other data sources to monitor for your application. You can then configure your application based on these recommendations.

Data preprocessing

CloudWatch Application Insights continuously analyzes the data sources being monitored across the application resources to discover metric anomalies and log errors (observations).

Intelligent problem detection

The CloudWatch Application Insights engine detects problems in your application by correlating observations using classification algorithms and built-in rules. To assist in troubleshooting, it creates automated CloudWatch dashboards, which include contextual information about the problems.

Alert and action

When CloudWatch Application Insights detects a problem with your application, it generates CloudWatch Events to notify you of the problem. See [Application Insights CloudWatch Events for detected problems] for more information about how to set up these Events. Additionally, you can [configure Amazon SNS notifications] to receive alerts for detected problems.

Example scenario

You have an ASP .NET application that is backed by a SQL Server database. Suddenly, your database begins to malfunction because of high memory pressure. This leads to application performance degradation and possibly HTTP 500 errors in your web servers and load balancer.

With CloudWatch Application Insights and its intelligent analytics, you can identify the application layer that is causing the problem by checking the dynamically created dashboard that shows the related metrics and log file snippets. In this case, the problem might be at the SQL database layer.

Data retention

CloudWatch Application Insights retains problems for 55 days and observations for 60 days.

Quotas

For default quotas for CloudWatch Application Insights, see [Amazon CloudWatch Application Insights endpoints and quotas]. Unless otherwise noted, each quota is per Amazon Region. Contact [Amazon Support] to request an increase in your service quota. Many services contain quotas that cannot be changed. For more information about the quotas for a specific service, see the documentation for that service.

Amazon Systems Manager (SSM) packages used by CloudWatch Application Insights

The packages listed in this section are used by Application Insights, and can be independently managed and deployed with Amazon Systems Manager Distributor. For more information about SSM Distributor, see [Amazon Systems Manager Distributor] in the *Amazon Systems Manager User Guide*.

Packages:

- AWSObservabilityExporter-JMXExporterInstallAndConfigure
- AWSObservabilityExporter-SAP-HANADBExporterInstallAndConfigure
- AWSObservabilityExporter-HAClusterExporterInstallAndConfigure
- AWSObservabilityExporter-SAP-SAPHostExporterInstallAndConfigure
- AWSObservabilityExporter-SQLExporterInstallAndConfigure

`AWSObservabilityExporter-JMXExporterInstallAndConfigure`

You can retrieve workload-specific Java metrics from Prometheus JMX exporter for Application Insights to configure and monitor alarms. In the Application Insights console, on the **Manage monitoring** page, select **JAVA application** from the **Application tier** dropdown. Then under **JAVA Prometheus exporter configuration**, select your **Collection method** and **JMX port number**.

To use Amazon Systems Manager Distributor to package, install, and configure the Amazon-provided Prometheus JMX exporter package independently of Application Insights, complete the following steps.

Prerequisites for using the Prometheus JMX exporter SSM package

- SSM agent version 2.3.1550.0 or later installed
- The JAVA_HOME environment variable is set

Install and configure the `AWSObservabilityExporter-JMXExporterInstallAndConfigure` package

The `AWSObservabilityExporter-JMXExporterInstallAndConfigure` package is an SSM Distributor package that you can use to install and configure Prometheus JMX Exporter. When Java metrics are sent by the Prometheus JMX exporter, the CloudWatch agent can be configured to retrieve the metrics for the CloudWatch service.

1. Based on your preferences, prepare the Prometheus JMX exporter YAML configuration file located in the Prometheus GitHub repository. Use the example configuration and option descriptions to guide you.
2. Copy the Prometheus JMX exporter YAML configuration file encoded as Base64 to a new SSM parameter in SSM Parameter Store.

3. Navigate to the SSM Distributor console and open the **Owned by Amazon** tab. Select **AWSObservabilityExporter-JMXExporterInstallAndConfigure** and choose **Install one time**.

4. Update the SSM parameter you created in the first step by replacing "Additional Arguments" with the following:

```
{
  "SSM_EXPORTER_CONFIGURATION": "{{ssm:<SSM_PARAMETER_STORE_NAME>}}",
  "SSM_EXPOSITION_PORT": "9404"
}
```

> **Note**
>
> Port 9404 is the default port used to send Prometheus JMX metrics. You can update this port.

Example: Configure CloudWatch agent to retrieve Java metrics

1. Install the Prometheus JMX exporter, as described in the previous procedure. Then verify that it is correctly installed on your instance by checking the port status.

   Successful installation on Windows instance example

```
PS C:\> curl http://localhost:9404 (http://localhost:9404/)
StatusCode : 200
StatusDescription : OK
Content : # HELP jvm_info JVM version info
```

   Successful installation on Linux instance example

```
$ curl localhost:9404
# HELP jmx_config_reload_failure_total Number of times configuration have failed to
 be reloaded.
# TYPE jmx_config_reload_failure_total counter
jmx_config_reload_failure_total 0.0
```

2. Create the Prometheus service discovery YAML file. The following example service discovery file performs the following:

- Specifies the Prometheus JMX exporter host port as `localhost: 9404`.
- Attaches labels (`Application`, `ComponentName`, and `InstanceId`) to the metrics, which can be set as CloudWatch metric dimensions.

```
$ cat prometheus_sd_jmx.yaml
- targets:
  - 127.0.0.1:9404
  labels:
    Application: myApp
    ComponentName: arn-cn:aws:elasticloadbalancing:cn-
north-1:123456789012:loadbalancer/app/sampl-Appli-MMZW8E3GH4H2/aac36d7fea2a6e5b
    InstanceId: i-12345678901234567
```

3. Create the Prometheus JMX exporter configuration YAML file. The following example configuration file specifies the following:

   - The metrics retrieval job interval and timeout period.
   - The metrics retrieval jobs (`jmx` and `sap`), also known as scraping, which include the job name, maximum time series returned at a time, and service discovery file path.

```
$ cat prometheus.yaml
global:
  scrape_interval: 1m
  scrape_timeout: 10s
scrape_configs:
  - job_name: jmx
    sample_limit: 10000
    file_sd_configs:
      - files: ["/tmp/prometheus_sd_jmx.yaml"]
  - job_name: sap
    sample_limit: 10000
    file_sd_configs:
      - files: ["/tmp/prometheus_sd_sap.yaml"]
```

4. Verify that the CloudWatch agent is installed on your Amazon EC2 instance and that the version is 1.247346.1b249759 or later. To install the CloudWatch agent on your EC2 instance, see Installing the CloudWatch Agent. To verify the version, see Finding information about CloudWatch agent versions.

5. Configure the CloudWatch agent. For more information about how to configure the CloudWatch agent configuration file, see [Manually create or edit the CloudWatch agent configuration file](). The following example CloudWatch agent configuration file performs the following:

   - Specifies the Prometheus JMX exporter configuration file path.
   - Specifies the target log group to which to publish EMF metric logs.
   - Specifies two sets of dimensions for each metric name.
   - Sends 8 (4 metric names * 2 sets of dimensions per metric name) CloudWatch metrics.

```
{
   "logs":{
      "logs_collected":{
         ....
      },
      "metrics_collected":{
         "prometheus":{
            "cluster_name":"prometheus-test-cluster",
            "log_group_name":"prometheus-test",
            "prometheus_config_path":"/tmp/prometheus.yaml",
            "emf_processor":{
               "metric_declaration_dedup":true,
               "metric_namespace":"CWAgent",
               "metric_unit":{
                  "jvm_threads_current":"Count",
                  "jvm_gc_collection_seconds_sum":"Second",
                  "jvm_memory_bytes_used":"Bytes"
               },
               "metric_declaration":[
                  {
                     "source_labels":[
                        "job"
                     ],
                     "label_matcher":"^jmx$",
                     "dimensions":[
                        [
                           "InstanceId",
                           "ComponentName"
                        ],
                        [
                           "ComponentName"
```

```
                        ]
                    ],
                    "metric_selectors":[
                        "^java_lang_threading_threadcount$",
                        "^java_lang_memory_heapmemoryusage_used$",
                        "^java_lang_memory_heapmemoryusage_committed$"
                    ]
                }
            ]
        }
      }
    }
  },
  "metrics":{
    ....
  }
}
```

AWSObservabilityExporter-SAP-HANADBExporterInstallAndConfigure

You can retrieve workload-specific SAP HANA metrics from Prometheus HANA database exporter for Application Insights to configure and monitor alarms. For more information, see Set up your SAP HANA database for monitoring in this guide.

To use Amazon Systems Manager Distributor to package, install, and configure the Amazon-provided Prometheus HANA database exporter package independently of Application Insights, complete the following steps.

Prerequisites for using the Prometheus HANA database exporter SSM package

- SSM agent version 2.3.1550.0 or later installed
- SAP HANA database
- Linux operating system (SUSE Linux, RedHat Linux)
- A secret with SAP HANA database monitoring credentials, using Amazon Secrets Manager. Create a secret using the key/value pairs format, specify the key username, and enter the database user for the value. Add a second key password, and then enter the password for the value. For more information about how to create secrets, see Create a secret in the *Amazon Secrets Manager User Guide*. The secret must be formatted as follows:

```
{
```

```
  "username": "<database_user>",
  "password": "<database_password>"
}
```

Install and configure the `AWSObservabilityExporter-SAP-HANADBExporterInstallAndConfigure` package

The `AWSObservabilityExporter-SAP-HANADBExporterInstallAndConfigure` package is an SSM Distributor package that you can use to install and configure Prometheus HANA database Exporter. When HANA database metrics are sent by the Prometheus HANA database exporter, the CloudWatch agent can be configured to retrieve the metrics for the CloudWatch service.

1. Create an SSM parameter in SSM Parameter Store to store the Exporter configurations. The following is an example parameter value.

```
{\"exposition_port\":9668,\"multi_tenant\":true,\"timeout\":600,\"hana\":{\"host\":
\"localhost\",\"port\":30013,\"aws_secret_name\":\"HANA_DB_CREDS\",\"scale_out_mode
\":true}}
```

> **Note**
>
> In this example, the export runs only on the Amazon EC2 instance with the active SYSTEM database, and it will remain idle on the other EC2 instances in order to avoid duplicate metrics. The exporter can retrieve all of the database tenant information from the SYSTEM database.

2. Create an SSM parameter in SSM Parameter Store to store the Exporter metrics queries. The package can accept more than one metrics parameter. Each parameter must have a valid JSON object format. The following is an example parameter value:

```
{\"SELECT MAX(TIMESTAMP) TIMESTAMP, HOST, MEASURED_ELEMENT_NAME CORE,
 SUM(MAP(CAPTION, 'User Time', TO_NUMBER(VALUE), 0)) USER_PCT, SUM(MAP(CAPTION,
 'System Time', TO_NUMBER(VALUE), 0)) SYSTEM_PCT, SUM(MAP(CAPTION, 'Wait
 Time', TO_NUMBER(VALUE), 0)) WAITIO_PCT, SUM(MAP(CAPTION, 'Idle Time', 0,
 TO_NUMBER(VALUE))) BUSY_PCT, SUM(MAP(CAPTION, 'Idle Time', TO_NUMBER(VALUE), 0))
 IDLE_PCT FROM sys.M_HOST_AGENT_METRICS WHERE MEASURED_ELEMENT_TYPE = 'Processor'
 GROUP BY HOST, MEASURED_ELEMENT_NAME;\":{\"enabled\":true,\"metrics\":[{\"name\":
\"hanadb_cpu_user\",\"description\":\"Percentage of CPU time spent by HANA DB in user
 space, over the last minute (in seconds)\",\"labels\":[\"HOST\",\"CORE\"],\"value\":
```

```
\"USER_PCT\",\"unit\":\"percent\",\"type\":\"gauge\"},{\"name\":\"hanadb_cpu_system
\",\"description\":\"Percentage of CPU time spent by HANA DB in Kernel space,
 over the last minute (in seconds)\",\"labels\":[\"HOST\",\"CORE\"],\"value\":
\"SYSTEM_PCT\",\"unit\":\"percent\",\"type\":\"gauge\"},{\"name\":\"hanadb_cpu_waitio
\",\"description\":\"Percentage of CPU time spent by HANA DB in IO mode, over the
 last minute (in seconds)\",\"labels\":[\"HOST\",\"CORE\"],\"value\":\"WAITIO_PCT\",
\"unit\":\"percent\",\"type\":\"gauge\"},{\"name\":\"hanadb_cpu_busy\",\"description
\":\"Percentage of CPU time spent by HANA DB, over the last minute (in seconds)\",
\"labels\":[\"HOST\",\"CORE\"],\"value\":\"BUSY_PCT\",\"unit\":\"percent\",\"type\":
\"gauge\"},{\"name\":\"hanadb_cpu_idle\",\"description\":\"Percentage of CPU time not
 spent by HANA DB, over the last minute (in seconds)\",\"labels\":[\"HOST\",\"CORE
\"],\"value\":\"IDLE_PCT\",\"unit\":\"percent\",\"type\":\"gauge\"}]}}
```

For more information about metrics queries, see the `SUSE / hanadb_exporter` repo on GitHub.

3. Navigate to the SSM Distributor console and open the **Owned by Amazon** tab. Select **AWSObservabilityExporter-SAP-HANADBExporterInstallAndConfigure*** and choose **Install one time**.
4. Update the SSM parameter you created in the first step by replacing "Additional Arguments" with the following:

```
{
  "SSM_EXPORTER_CONFIG": "{{ssm:<*SSM_CONFIGURATIONS_PARAMETER_STORE_NAME>*}}",
  "SSM_SID": "<SAP_DATABASE_SID>",
  "SSM_EXPORTER_METRICS_1": "{{ssm:<SSM_FIRST_METRICS_PARAMETER_STORE_NAME>}}",
  "SSM_EXPORTER_METRICS_2": "{{ssm:<SSM_SECOND_METRICS_PARAMETER_STORE_NAME>}}"
}
```

5. Select the Amazon EC2 instances with SAP HANA database, and choose **Run**.

`AWSObservabilityExporter-HAClusterExporterInstallAndConfigure`

You can retrieve workload-specific High Availability (HA) cluster metrics from Prometheus HANA cluster exporter for Application Insights to configure and monitor alarms for an SAP HANA database High Availability setup. For more information, see Set up your SAP HANA database for monitoring in this guide.

To use Amazon Systems Manager Distributor to package, install, and configure the Amazon-provided Prometheus HA cluster exporter package independently of Application Insights, complete the following steps.

Prerequisites for using the Prometheus HA cluster exporter SSM package

- SSM agent version 2.3.1550.0 or later installed
- HA cluster for Pacemaker, Corosync, SBD, and DRBD
- Linux operating system (SUSE Linux, RedHat Linux)

Install and configure the `AWSObservabilityExporter-HAClusterExporterInstallAndConfigure` package

The `AWSObservabilityExporter-HAClusterExporterInstallAndConfigure` package is an SSM Distributor package that you can use to install and configure Prometheus HA Cluster Exporter. When cluster metrics are sent by the Prometheus HANA database exporter, the CloudWatch agent can be configured to retrieve the metrics for the CloudWatch service.

1. Create an SSM parameter in SSM Parameter Store to store the Exporter configurations in JSON format. The following is an example parameter value.

   ```
   {\"port\":\"9664\",\"address\":\"0.0.0.0\",\"log-level\":\"info\",\"crm-mon-path
   \":\"/usr/sbin/crm_mon\",\"cibadmin-path\":\"/usr/sbin/cibadmin\",\"corosync-
   cfgtoolpath-path\":\"/usr/sbin/corosync-cfgtool\",\"corosync-quorumtool-path\":\"/
   usr/sbin/corosync-quorumtool\",\"sbd-path\":\"/usr/sbin/sbd\",\"sbd-config-path\":
   \"/etc/sysconfig/sbd\",\"drbdsetup-path\":\"/sbin/drbdsetup\",\"enable-timestamps
   \":false}
   ```

   For more information about the exporter configurations, see the `ClusterLabs / ha_cluster_exporter` repo on GitHub.
2. Navigate to the SSM Distributor console and open the **Owned by Amazon** tab. Select **AWSObservabilityExporter-HAClusterExporterInstallAndConfigure*** and choose **Install one time**.
3. Update the SSM parameter you created in the first step by replacing "Additional Arguments" with the following:

   ```
   {
     "SSM_EXPORTER_CONFIG": "{{ssm:<*SSM_CONFIGURATIONS_PARAMETER_STORE_NAME>*}}"
   }
   ```
4. Select the Amazon EC2 instances with SAP HANA database, and choose **Run**.

`AWSObservabilityExporter-SAP-SAPHostExporterInstallAndConfigure`

You can retrieve workload-specific SAP NetWeaver metrics from Prometheus SAP host exporter for Application Insights to configure and monitor alarms for SAP NetWeaver Distributed and High Availability deployments.

To use Amazon Systems Manager Distributor to package, install, and configure the SAP host exporter package independently of Application Insights, complete the following steps.

Prerequisites for using the Prometheus SAP host exporter SSM package

- SSM agent version 2.3.1550.0 or later installed
- SAP NetWeaver application servers
- Linux operating system (SUSE Linux, RedHat Linux)

Install and configure the `AWSObservabilityExporter-SAP-SAPHostExporterInstallAndConfigure` package

The `AWSObservabilityExporter-SAP-SAPHostExporterInstallAndConfigure` package is an SSM Distributor package that you can use to install and configure SAP NetWeaver Prometheus metrics exporter. When SAP NetWeaver metrics are sent by the Prometheus exporter, the CloudWatch agent can be configured to retrieve the metrics for the CloudWatch service.

1. Create an SSM parameter in SSM Parameter Store to store the Exporter configurations in JSON format. The following is an example parameter value.

   ```
   {\"address\":\"0.0.0.0\",\"port\":\"9680\",\"log-level\":\"info\",\"is-HA\":false}
   ```

   - **address**

     The target address to which to send the Prometheus metrics. The default value is `localhost`.

   - **port**

     The target port to which to send the Prometheus metrics. The default value is `9680`.

   - **is-HA**

     `true` for SAP NetWeaver High Availability deployments. For all other deployments the value is `false`.

2. Navigate to the SSM Distributor console and open the **Owned by Amazon** tab. Select **AWSObservabilityExporter-SAP-SAPHostExporterInstallAndConfigure** and choose **Install one time**.

3. Update the SSM parameter you created in the first step by replacing "Additional Arguments" with the following:

```
{
  "SSM_EXPORTER_CONFIG": "{{ssm:<SSM_CONFIGURATIONS_PARAMETER_STORE_NAME>}}",
  "SSM_SID": "<SAP_DATABASE_SID>",
  "SSM_INSTANCES_NUM": "<instances_number seperated by comma>"
}
```

Example

```
{
  "SSM_EXPORTER_CONFIG": "{{ssm:exporter_config_paramter}}",
  "SSM_INSTANCES_NUM": "11,12,10",
  "SSM_SID": "PR1"
}
```

4. Select the Amazon EC2 instances with SAP NetWeaver applications, and choose **Run**.

> **Note**
>
> The Prometheus exporter services the SAP NetWeaver metrics on a local endpoint. The local endpoint can be accessed by only the operating system users on the Amazon EC2 instance. Therefore, after the exporter package is installed, the metrics are available to all of the operating system users. The default local endpoint is `localhost:9680/metrics`.

`AWSObservabilityExporter-SQLExporterInstallAndConfigure`

You can retrieve workload-specific SQL Server metrics from Prometheus SQL exporter for Application Insights to monitor key metrics.

To use Amazon Systems Manager Distributor to package, install, and configure the SQL exporter package independently of Application Insights, complete the following steps.

Prerequisites for using the Prometheus SQL exporter SSM package

- SSM agent version 2.3.1550.0 or later installed
- Amazon EC2 instance running SQL Server on Windows with SQL Server user authentication enabled.
- A SQL Server user with the following permissions:

```
GRANT VIEW ANY DEFINITION TO
```

```
GRANT VIEW SERVER STATE TO
```

- A secret containing the database connection string using Amazon Secrets Manager. For more information about how to create secrets, see Create a secret in the *Amazon Secrets Manager User Guide*. The secret must be formatted as follows:

```
{
  "data_source_name":"sqlserver://<username>:<password>@localhost:1433"
}
```

> **Note**
>
> If the password or username contains special characters, you must percent encode the special characters to ensure a successful connection to the database.

Install and configure the `AWSObservabilityExporter-SQLExporterInstallAndConfigure` package

The `AWSObservabilityExporter-SQLExporterInstallAndConfigure` package is an SSM Distributor package that you can use to install and configure SQL Prometheus metrics exporter. When metrics are sent by the Prometheus exporter, the CloudWatch agent can be configured to retrieve the metrics for the CloudWatch service.

1. Based on your preferences, prepare the SQL Exporter YAML configuration. The following sample configuration has a single metric configured. Use the example configuration to update the configuration with additional metrics or create your own configuration.

```
---
```

```
global:
  scrape_timeout_offset: 500ms
  min_interval: 0s
  max_connections: 3
  max_idle_connections: 3
target:
  aws_secret_name: <SECRET_NAME>
  collectors:
    - mssql_standard
collectors:
  - collector_name: mssql_standard
    metrics:
    - metric_name: mssql_batch_requests
      type: counter
      help: 'Number of command batches received.'
      values: [cntr_value]
      query: |
        SELECT cntr_value
        FROM sys.dm_os_performance_counters WITH (NOLOCK)
        WHERE counter_name = 'Batch Requests/sec'
```

2. Copy the Prometheus SQL exporter YAML configuration file encoded as Base64 to a new SSM parameter in SSM Parameter Store.
3. Navigate to the SSM Distributor console and open the **Owned by Amazon** tab. Select **AWSObservabilityExporter-SQLExporterInstallAndConfigure** and choose **Install one time**.
4. Replace the "Additional Arguments" with the following information. The `SSM_PARAMETER_NAME` is the name of the parameter you created in Step 2.

```
{
  "SSM_EXPORTER_CONFIGURATION":
    "{{ssm:<SSM_PARAMETER_STORE_NAME>}}",
    "SSM_PROMETHEUS_PORT": "9399",
    "SSM_WORKLOAD_NAME": "SQL"
}
```

5. Select the Amazon EC2 instance with the SQL Server database, then choose run.

Amazon Systems Manager (SSM) Documents used by CloudWatch Application Insights

Application Insights uses the SSM Documents listed in this section to define the actions that Amazon Systems Manager performs on your managed instances. These documents use the `Run Command` capability of Systems Manager to automate the tasks necessary for carrying out Application Insights monitoring capabilities. The run schedules for these documents are maintained by Application Insights and can't be altered.

For more information about SSM Documents, see [Amazon Systems Manager Documents] in the *Amazon Systems Manager User Guide*.

Documents managed by CloudWatch Application Insights

The following table lists the SSM documents that are managed by Application Insights.

Document name	Description	Run schedule
`AWSEC2-DetectWorkload`	Auto detects applications running in your application environment that can be set up to be monitored by Application Insights.	This document runs hourly in your application environment to get up-to-date application details.
`AWSEC2-CheckPerformanceCounterSets`	Checks whether Performance Counter namespaces are enabled on your Amazon EC2 Windows instances.	This document runs hourly in your application environment and only monitors Performance Counter metrics if the corresponding namespaces are enabled.
`AWSEC2-ApplicationInsightsCloudwatchAgentInstallAndConfigure`	Installs and configures CloudWatch Agent based on the monitoring configuration of your application components.	This document runs every 30 minutes to ensure that the CloudWatch Agent configuration is always accurate and up-to-date. The document also runs immediately after a change is made to your application monitoring setup

Document name	Description	Run schedule
		such as adding or removing metrics or updating log configurations.

Documents managed by Amazon Systems Manager

The following documents are used by CloudWatch Application Insights and managed by Systems Manager.

`AWS-ConfigureAWSPackage`

Application Insights uses this document to install and uninstall Prometheus exporter distributor packages, to collect workload specific metrics, and to enable comprehensive monitoring of workloads on customer Amazon EC2 instances. CloudWatch Application Insights installs the Prometheus exporter distributor packages only if the correlated target workload is running on your instance.

The following table lists the Prometheus exporter distributor packages and the correlated target workloads.

Prometheus exporter distributor package name	Target workload
`AWSObservabilityExporter-HAClusterExporterInstallAndConfigure`	SAP HANA HA
`AWSObservabilityExporter-JMXExporterInstallAndConfigure`	Java/JMX
`AWSObservabilityExporter-SAP-HANADBExporterInstallAndConfigure`	SAP HANA
`AWSObservabilityExporter-SAP-SAPHostExporterInstallAndConfigure`	NetWeaver

Prometheus exporter distributor package name	Target workload
`AWSObservabilityExporter-SQLExporterInstallAndConfigure`	SQL Server (Windows) and SAP ASE (Linux)

`AmazonCloudWatch-ManageAgent`

Application Insights uses this document to manage the status and configuration of CloudWatch Agent on your instances and to collect internal system level metrics and logs from Amazon EC2 instances across operating systems.

Prerequisites, IAM policies, and permissions needed to access CloudWatch Application Insights

To get started with CloudWatch Application Insights, verify that you have met the following prerequisites, have created an IAM policy, and have attached permissions if needed.

Topics

Prerequisites to configure an application for monitoring

You must complete the following prerequisites to configure an application with CloudWatch Application Insights:

- **Amazon Systems Manager enablement** – Install Systems Manager Agent (SSM Agent) on your Amazon EC2 instances, and enable the instances for SSM. For information about how to install the SSM Agent, see Setting up Amazon Systems Manager in the *Amazon Systems Manager User Guide*.
- **EC2 instance role** – You must attach the following Amazon EC2 instance roles to enable Systems Manager
  - You must attach the `AmazonSSMManagedInstanceCore` role to enable Systems Manager. For more information, see Amazon Systems Manager identity-based policy examples.

  - You must attach the `CloudWatchAgentServerPolicy` policy to enable instance metrics and logs to be emitted through CloudWatch. For more information, see [Create IAM roles and users for use with CloudWatch agent](#).

- **Amazon resource groups –** To onboard your applications to CloudWatch Application Insights, create a resource group that includes all of the associated Amazon resources used by your application stack. This includes application load balancers, Amazon EC2 instances running IIS and web front-end, .NET worker tiers, and SQL Server databases. For more information about application components and technology stacks supported by Application Insights, see Supported application components. CloudWatch Application Insights automatically includes Auto Scaling groups using the same tags or CloudFormation stacks as your resource group, because Auto Scaling groups are not supported by CloudFormation resource groups. For more information, see Getting Started with Amazon Resource Groups.
- **IAM permissions –** For users who don't have administrative access, you must create an Amazon Identity and Access Management (IAM) policy that allows Application Insights to create a service-linked role and attach it to the user's identity. For more information about how to create the IAM policy, see IAM policy for CloudWatch Application Insights.
- **Service-linked role –** Application Insights uses Amazon Identity and Access Management (IAM) service-linked roles. A service-linked role is created for you when you create your first Application Insights application in the Application Insights console. For more information, see Using service-linked roles for CloudWatch Application Insights.
- **Performance Counter metrics support for EC2 Windows instances –** To monitor Performance Counter metrics on your Amazon EC2 Windows instances, Performance Counters must be installed on the instances. For Performance Counter metrics and corresponding Performance Counter set names, see Performance Counter metrics. For more information about Performance Counters, see Performance Counters.
- **Amazon CloudWatch agent –** Application Insights installs and configures the CloudWatch agent. If you have CloudWatch agent installed, Application Insights retains your configuration. To avoid a merge conflict, remove the configuration of resources that you want to use in Application Insights from the existing CloudWatch agent configuration file. For more information, see Manually create or edit the CloudWatch agent configuration file.

IAM policy for CloudWatch Application Insights

To use CloudWatch Application Insights, you must create an Amazon Identity and Access Management (IAM) policy and attach it to your user, group, or role. For more information about users, groups, and roles, see IAM Identities (users, user groups, and roles). The IAM policy defines the user permissions.

To create an IAM policy using the console

To create an IAM policy using the IAM console, perform the following steps.

1. Go to the IAM console. In the left navigation pane, select **Policies**.
2. At the top of the page, select **Create policy**.
3. Select the **JSON** tab.
4. Copy and paste the following JSON document under the **JSON** tab.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "applicationinsights:*",
                "iam:CreateServiceLinkedRole",
                "iam:ListRoles",
                "resource-groups:ListGroups"
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
    ]
}
```

5. Select **Review Policy**.
6. Enter a **Name** for the policy, for example, “AppInsightsPolicy.” Optionally, enter a **Description**.
7. Select **Create Policy**.
8. In the left navigation pane, select **User groups**, **Users**, or **Roles**.
9. Select the name of the user group, user, or role to which you would like to attach the policy.
10. Select **Add permissions**.
11. Select **Attach existing policies directly**.
12. Search for the policy that you just created, and select the check box to the left of the policy name.
13. Select **Next: Review**.

14. Make sure that the correct policy is listed, and select **Add permissions**.
15. Make sure that you log in with the user associated with the policy that you just created when you use CloudWatch Application Insights.

To create an IAM policy using the Amazon CLI

To create an IAM policy using the Amazon CLI, run the [create-policy](https://docs.aws.amazon.com/cli/latest/reference/iam/create-policy.html) operation from the command line using the JSON document above as a file in your current folder.

To create an IAM policy using Amazon Tools for Windows PowerShell

To create an IAM policy using the Amazon Tools for Windows PowerShell, run the [New-IAMPolicy](https://docs.aws.amazon.com/powershell/latest/reference/items/New-IAMPolicy.html) cmdlt using the JSON document above as a file in your current folder.

IAM role permissions for account-based application onboarding

If you want to onboard all of the resources in your account, and you choose not to use the [Application Insights managed policy](#) for full access to Application Insights functionality, you must attach the following permissions to your IAM role so that Application Insights can discover all of the resources in your account:

```
"ec2:DescribeInstances"
"ec2:DescribeNatGateways"
"ec2:DescribeVolumes"
"ec2:DescribeVPCs"
"rds:DescribeDBInstances"
"rds:DescribeDBClusters"
"sqs:ListQueues"
"elasticloadbalancing:DescribeLoadBalancers"
"autoscaling:DescribeAutoScalingGroups"
"lambda:ListFunctions"
"dynamodb:ListTables"
"s3:ListAllMyBuckets"
"sns:ListTopics"
"states:ListStateMachines"
"apigateway:GET"
"ecs:ListClusters"
"ecs:DescribeTaskDefinition"
"ecs:ListServices"
"ecs:ListTasks"
"eks:ListClusters"
```

```
"eks:ListNodegroups"
"fsx:DescribeFileSystems"
"route53:ListHealthChecks"
"route53:ListHostedZones"
"route53:ListQueryLoggingConfigs"
"route53resolver:ListFirewallRuleGroups"
"route53resolver:ListFirewallRuleGroupAssociations"
"route53resolver:ListResolverEndpoints"
"route53resolver:ListResolverQueryLogConfigs"
"route53resolver:ListResolverQueryLogConfigAssociations"
"logs:DescribeLogGroups"
"resource-explorer:ListResources"
```

Set up application for monitoring using the Amazon Web Services Management Console

This section provides steps to set up, configure, and manage your CloudWatch Application Insights application using the console, the Amazon CLI, and Amazon Tools for Windows PowerShell.

Topics

- Set up, configure, and manage your application for monitoring from the CloudWatch console
- Set up, configure, and manage your application for monitoring using the command line
- Application Insights CloudWatch Events for detected problems
- Receive notifications for detected problems

Set up, configure, and manage your application for monitoring from the CloudWatch console

This section provides steps to set up, configure, and manage your application for monitoring from the CloudWatch console.

Console procedures

- Add and configure an application
- Enable Application Insights for Amazon ECS and Amazon EKS resource monitoring
- Disable monitoring for an application component
- Delete an application

Add and configure an application

Add and configure an application from the CloudWatch console

To get started with CloudWatch Application Insights from the CloudWatch console, perform the following steps.

1. **Start.** Open the CloudWatch console landing page. From the left navigation pane, under **Insights**, choose **Application Insights**. The page that opens shows the list of applications that are monitored with CloudWatch Application Insights, along with their monitoring status.
2. **Add an application.** To set up monitoring for your application, choose **Add an application**. When you choose **Add an application**, you are prompted to **Choose Application Type**.

    - **Resource group-based application**. When you select this option, you can choose which resource groups in this account to monitor. To use multiple applications on a component, you must use resource group-based monitoring.
    - **Account-based application**. When you select this option, you can monitor all of the resources in this account. If you want to monitor all of the resources in an account, we recommend this option over the resource group-based option because the application onboarding process is faster.

    > **Note**
    >
    > You can't combine resource group-based monitoring with account-based monitoring using Application Insights. In order to change the application type, you must delete all of the applications that are being monitored, and **Choose Application Type**.

    When you add your first application for monitoring, CloudWatch Application Insights creates a service-linked role in your account, which gives Application Insights permissions to call other Amazon services on your behalf. For more information about the service-linked role created in your account by Application Insights, see Using service-linked roles for CloudWatch Application Insights.
3. Resource-based application monitoring

    1. **Select an application or resource group.** On the **Specify application details** page, select the Amazon resource group that contains your application resources from the

dropdown list. These resources include front-end servers, load balancers, auto scaling groups, and database servers.

If you have not created a resource group for your application, you can create one by choosing **Create new resource group**. For more information about creating resource groups, see the *Amazon Resource Groups User Guide*.

2. **Notifications for problem insights.** To view and get notified when problems are detected for selected applications, choose Amazon SNS notifications or Systems Manager OpsCenter Opsitems.

   a. **Set up Amazon SNS notification (Recommended).** Choose **Select existing topic** or **Create new topic**.

   b. **Integrate with Amazon Systems Manager OpsCenter.** Under **Advanced Settings**, select the **Generate Systems Manager OpsCenter OpsItems for remedial actions** check box. To track the operations that are taken to resolve operational work items (OpsItems) that are related to your Amazon resources, provide the Amazon SNS topic ARN.

3. **Monitor CloudWatch Events**. Select the check box to integrate Application Insights monitoring with CloudWatch Events to get insights from Amazon EBS, Amazon EC2, Amazon CodeDeploy, Amazon ECS, Amazon Health APIs And Notifications, Amazon RDS, Amazon S3, and Amazon Step Functions.

4. **Tags — optional**. CloudWatch Application Insights supports both tag-based and CloudFormation-based resource groups (with the exception of Auto Scaling groups). For more information, see Working with Tag Editor.

5. Choose **Next**.

   An ARN is generated for the application in the following format.

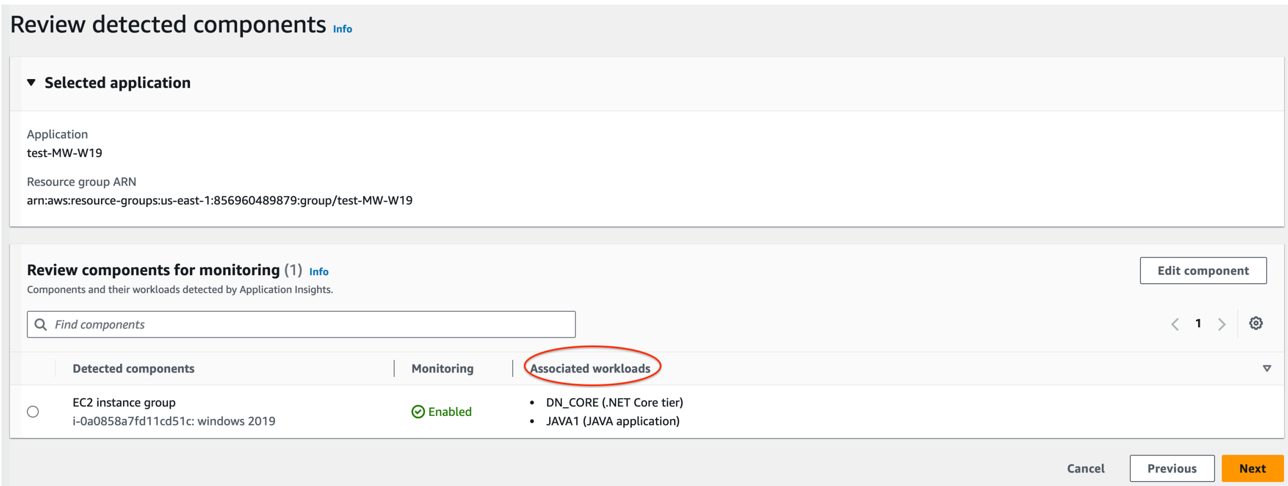
```
arn:partition:applicationinsights:region:account-id:application/resource-group/resource-group-name
```

   Example

```
arn-cn:aws:applicationinsights:cn-north-1:123456789012:application/resource-group/my-resource-group
```

6. On the **Review detected components** page, under **Review components for monitoring**, the table lists the detected components and their associated detected workloads.

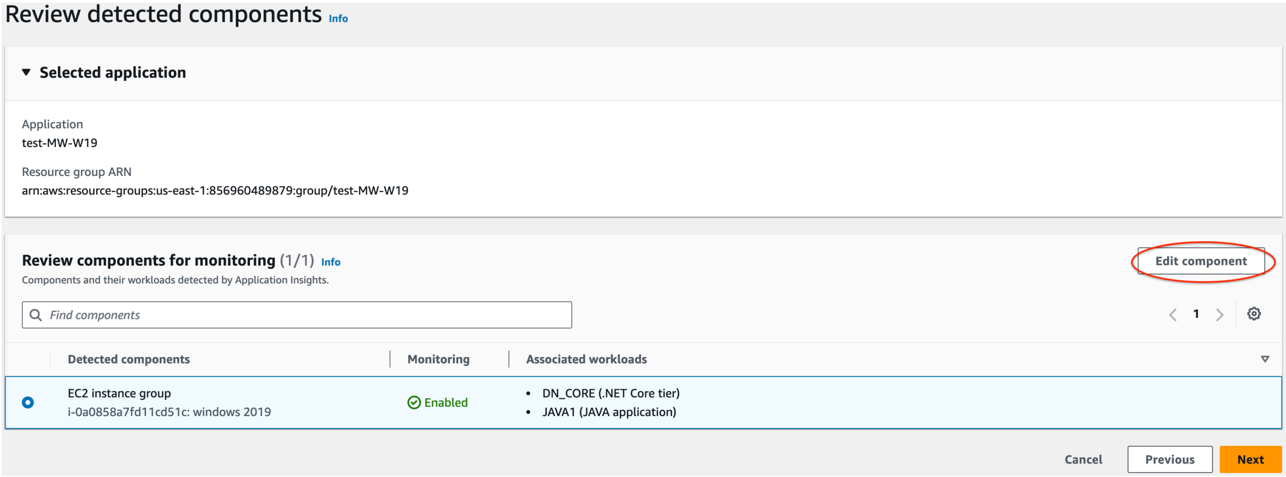
> **Note**
>
> For components that support multiple customized workloads, you can monitor up to five workloads for each component. These workloads will be monitored separately from the component.



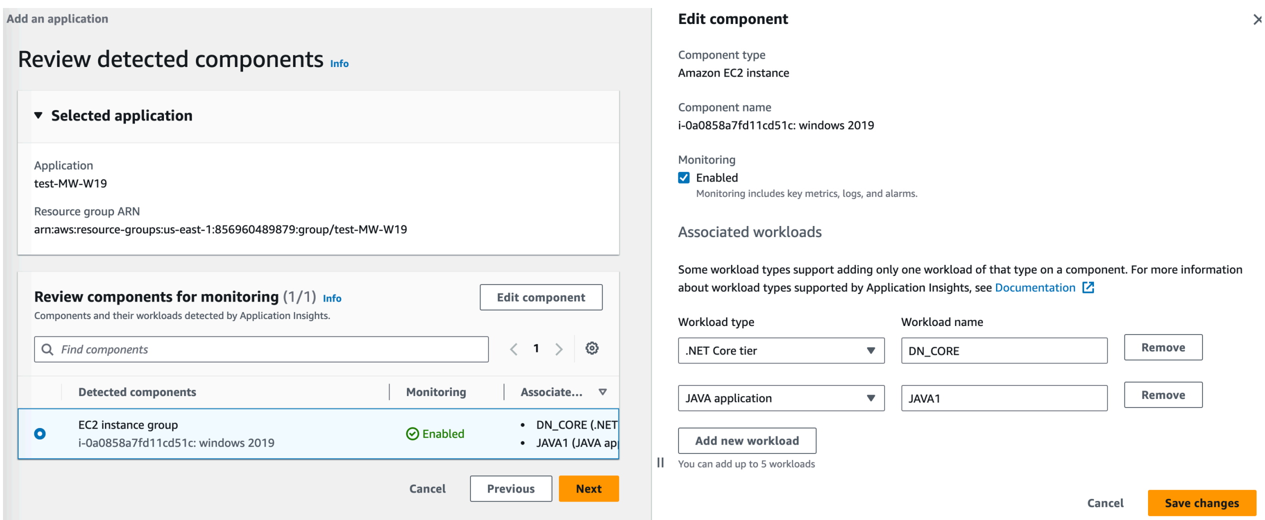
Under **Associated workloads**, there are several possible messages that appear if a workload is not listed.

- **Couldn't detect workloads** – An issue occurred when trying to detect workloads. Make sure that you have completed the Prerequisites to configure an application for monitoring. If you need to add workloads, choose **Edit component**.
- **No workloads detected** – We didn't detect any workloads. You may need to add workloads. To do so, choose **Edit component**.
- **Not applicable** – The component doesn't support customized workloads and will be monitored with default metrics, alarms, and logs. You can't add workloads to these components.

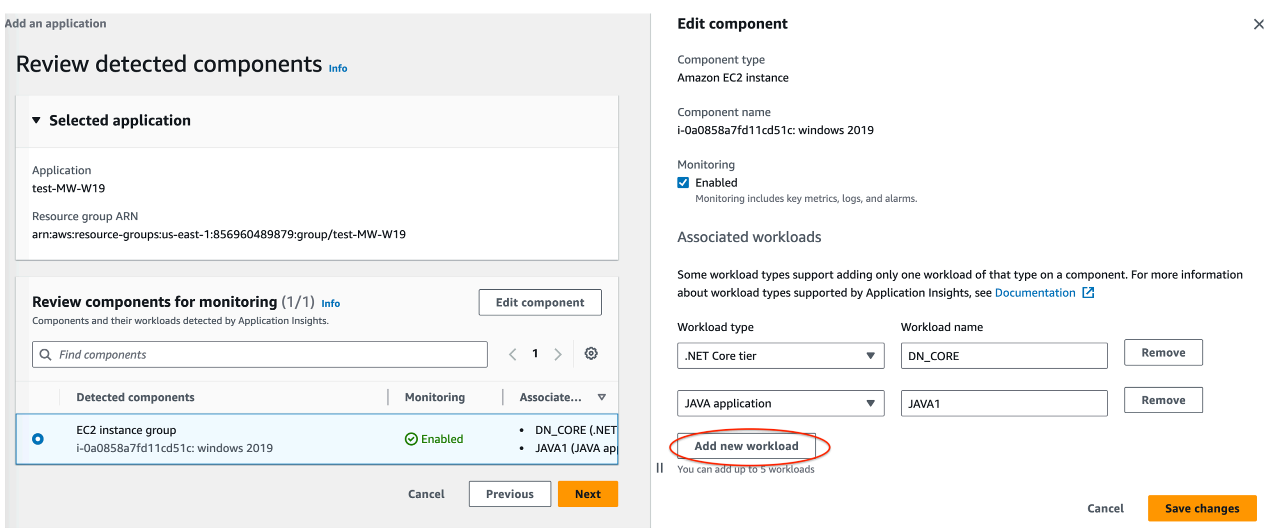
7. To edit a component, select a component, and then choose **Edit component**. A side panel opens with workloads detected on the component. In this panel, you can edit the component details and add new workloads.



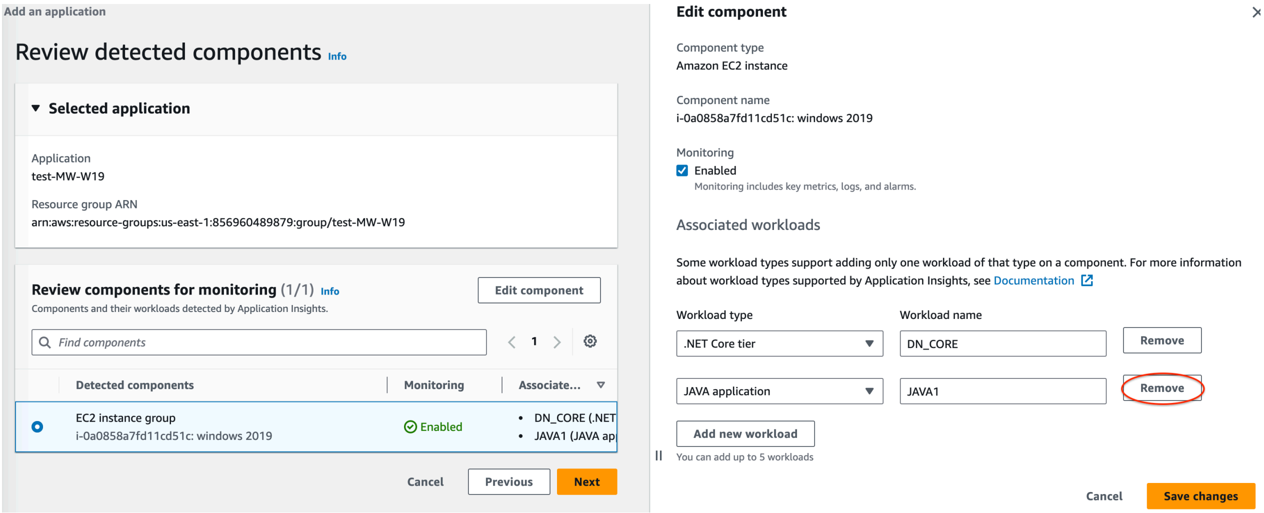
- To edit the workload type or name, use the dropdown list.



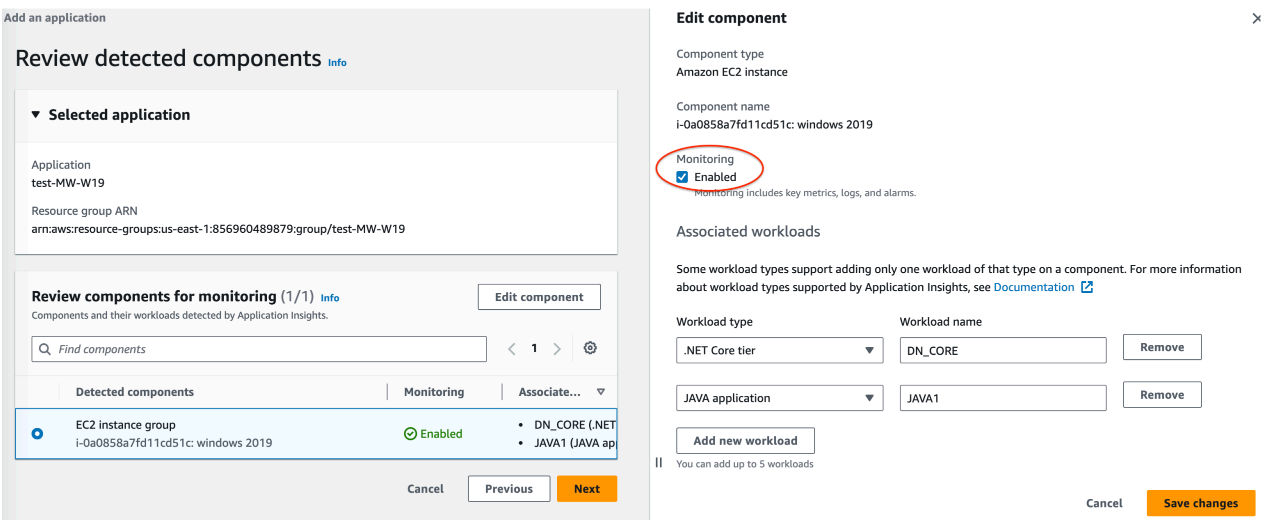
- To add a workload to the component, choose **Add new workload**.



  - If **Add new workload** doesn't appear, this component doesn't support multiple workloads.
  - If the **Associated workloads** heading doesn't appear, this component doesn't support customized workloads.
- To remove a workload, choose **Remove** next to the workload that you want to remove from monitoring.



- To disable monitoring for the entire component, clear the **Monitoring** check box.



- When you are done editing the component, choose **Save changes** in the lower right corner. Any changes to workloads for a component are visible on the **Review components for monitoring** table under **Associated workloads**.

8. On the **Review detected components** page, choose **Next**.
9. The **Specify component details** page includes all components with customizable associated workloads from the previous step.

> **Note**
>
> If a component header has an *optional* tag, additional details for the workloads in that component are optional.

If a component doesn't appear on this page, the component doesn't have any additional details that can be specified in this step.

10. Choose **Next**.

11. On the **Review and submit** page, review all monitored component and workload details.

12. Choose **Submit**.

Account-based application monitoring

1. **Application name**. Enter a name for your account-based application.
2. **Automated monitoring of new resources**. By default, Application Insights uses recommended settings to configure monitoring for resource components that are added to your account after you onboard the application. You can exclude monitoring for resources added after onboarding your application by clearing the check box.
3. **Monitor CloudWatch Events**. Select the check box to integrate Application Insights monitoring with CloudWatch Events to get insights from Amazon EBS, Amazon EC2, Amazon CodeDeploy, Amazon ECS, Amazon Health APIs And Notifications, Amazon RDS, Amazon S3, and Amazon Step Functions.
4. **Integrate with Amazon Systems Manager OpsCenter.** To view and get notified when problems are detected for selected applications, select the **Generate Systems Manager OpsCenter OpsItems for remedial actions** check box. To track the operations that are taken to resolve operational work items (OpsItems) that are related to your Amazon resources, provide the SNS topic ARN.
5. **Tags — optional**. CloudWatch Application Insights supports both tag-based and CloudFormation-based resource groups (with the exception of Auto Scaling groups). For more information, see [Working with Tag Editor](#).
6. **Discovered resources**. All of the resources discovered in your account are added to this list. If Application Insights is unable to discover all of the resources in your account,

an error message appears at the top of the page. This message includes a link to the documentation for how to add the required permissions.

7. Choose **Next**.

   An ARN is generated for the application in the following format.

   ```
   arn:partition:applicationinsights:region:account-id:application/
   TBD/application-name
   ```

   Example

   ```
   arn-cn:aws:applicationinsights:cn-north-1:123456789012:application/TBD/my-
   application
   ```

4. After you submit your application monitoring configuration, you will be taken to the details page for the application, where you can view the **Application summary**, the list of **Monitored components** and **Unmonitored components**, and, by selecting the tabs next to **Components**, the **Configuration history**, **Log patterns**, and any **Tags** that you have applied.

   To view insights for the application, choose **View Insights**.

   You can update your selections for CloudWatch Events monitoring and integration with Amazon Systems Manager OpsCenter by choosing **Edit**.

   Under **Components**, you can select the **Actions** menu to Create, Modify, or Ungroup an instance group.

   You can manage monitoring for components, including application tier, log groups, event logs, metrics, and custom alarms, by selecting the bullet next to a component and choosing **Manage monitoring**.

Enable Application Insights for Amazon ECS and Amazon EKS resource monitoring

You can enable Application Insights to monitor containerized applications and microservices from the Container Insights console. Application Insights supports monitoring for the following resources:

- Amazon ECS clusters
- Amazon ECS services

- Amazon ECS tasks
- Amazon EKS clusters

When Application Insights is enabled, it provides recommended metrics and logs, detects potential problems, generates CloudWatch Events, and creates automatic dashboards for your containerized applications and microservices.

You can enable Application Insights for containerized resources from the Container Insights or Application Insights consoles.

Enable Application Insights from the Container Insights console

From the Container Insights console, on the Container Insights **Performance monitoring** dashboard, choose **Auto-configure Application Insights**. When Application Insights is enabled, it displays details about detected problems.

Enable Application Insights from the Application Insights console

When ECS clusters appear in the component list, Application Insights automatically enables additional container monitoring with Container Insights.

For EKS clusters, you can enable additional monitoring with Container Insights to provide diagnostics information, such as container restart failures, to help you isolate and resolve problems. Additional steps are required to set up Container Insights for EKS. For information, see Setting up Container Insights on Amazon EKS and Kubernetes for steps to set up Container Insights on EKS.

Additional monitoring for EKS with Container Insights is supported on Linux instances with EKS.

For more information about Container Insights support for ECS and EKS clusters, see Container Insights.

Disable monitoring for an application component

To disable monitoring for an application component, from the application details page, select the component for which you want to disable monitoring. Choose **Actions**, and then **Remove from monitoring**.

Delete an application

To delete an application, from the CloudWatch dashboard, on the left navigation pane, choose **Application Insights** under **Insights**. Select the application that you want to delete. Under **Actions**, choose **Delete application**. This deletes monitoring and deletes all of the saved monitors for application components. The application resources are not deleted.

Set up, configure, and manage your application for monitoring using the command line

This section provides steps for setting up, configuring, and managing your application for monitoring using the Amazon CLI and Amazon Tools for Windows PowerShell.

Command line procedures

- Add and manage an application
- Manage and update monitoring
- Configure monitoring for SQL Always On Availability Groups
- Configure monitoring for MySQL RDS
- Configure monitoring for MySQL EC2
- Configure monitoring for PostgreSQL RDS
- Configure monitoring for PostgreSQL EC2
- Configure monitoring for Oracle RDS
- Configure monitoring for Oracle EC2

Add and manage an application

You can add, get information about, manage, and configure your Application Insights application using the command line.

Topics

- Add an application
- Describe an application
- List components in an application
- Describe a component
- Group similar resources into a custom component
- Ungroup a custom component
- Update an application

- Update a custom component

Add an application

Add an application using the Amazon CLI

To use the Amazon CLI to add an application for your resource group called `my-resource-group`, with OpsCenter enabled to deliver the created opsItem to the SNS topic ARN `arn-cn:aws:sns:cn-north-1:123456789012:MyTopic`, use the following command.

```
aws application-insights create-application --resource-group-name my-resource-group --ops-center-enabled --ops-item-sns-topic-arn arn-cn:aws:sns:cn-north-1:123456789012:MyTopic
```

Add an application using Amazon Tools for Windows PowerShell

To use Amazon Tools for Windows PowerShell to add an application for your resource group called `my-resource-group` with OpsCenter enabled to deliver the created opsItem to the SNS topic ARN `arn:aws:sns:cn-north-1:123456789012:MyTopic`, use the following command.

```
New-CWAIApplication -ResourceGroupName my-resource-group -OpsCenterEnabled true -OpsItemSNSTopicArn arn-cn:aws:sns:cn-north-1:123456789012:MyTopic
```

Describe an application

Describe an application using the Amazon CLI

To use the Amazon CLI to describe an application created on a resource group called `my-resource-group`, use the following command.

```
aws application-insights describe-application --resource-group-name my-resource-group
```

Describe an application using Amazon Tools for Windows PowerShell

To use the Amazon Tools for Windows PowerShell to describe an application created on a resource group called `my-resource-group`, use the following command.

```
Get-CWAIApplication -ResourceGroupName my-resource-group
```

List components in an application

List components in an application using the Amazon CLI

To use the Amazon CLI to list the components created on a resource group called `my-resource-group`, use the following command.

```
aws application-insights list-components --resource-group-name my-resource-group
```

List components in an application using Amazon Tools for Windows PowerShell

To use the Amazon Tools for Windows PowerShell to list the components created on a resource group called `my-resource-group`, use the following command.

```
Get-CWAIComponentList -ResourceGroupName my-resource-group
```

Describe a component

Describe a component using the Amazon CLI

You can use the following Amazon CLI command to describe a component called `my-component` that belongs to an application created on a resource group called `my-resource-group`.

```
aws application-insights describe-component --resource-group-name my-resource-group --
component-name my-component
```

Describe a component using Amazon Tools for Windows PowerShell

You can use the following Amazon Tools for Windows PowerShell command to describe a component called `my-component` that belongs to an application created on a resource group called `my-resource-group`.

```
Get-CWAIComponent -ComponentName my-component -ResourceGroupName my-resource-group
```

Group similar resources into a custom component

We recommend grouping similar resources, such as .NET web server instances, into custom components for easier onboarding and better monitoring and insights. Currently, CloudWatch Application Insights supports custom groups for EC2 instances.

To group resources into a custom component using the Amazon CLI

To use the Amazon CLI to group three instances (`arn-cn:aws:ec2:cn-north-1:123456789012:instance/i-11111`, `arn-cn:aws:ec2:cn-north-1:123456789012:instance/i-22222`, and `arn-cn:aws:ec2:cn-north-1:123456789012:instance/i-33333`) together into a custom component called `my-component` for an application created for the resource group called `my-resource-group`, use the following command.

```
aws application-insights create-component --resource-group-name my-resource-group --component-name my-component --resource-list arn:aws-cn:ec2:cn-north-1:123456789012:instance/i-11111 arn:aws-cn:ec2:cn-north-1:123456789012:instance/i-22222 arn:aws-cn:ec2:cn-north-1:123456789012:instance/i-33333
```

To group resources into a custom component using Amazon Tools for Windows PowerShell

To use Amazon Tools for Windows PowerShell to group three instances (`arn:aws-cn:ec2:cn-north-1:123456789012:instance/i-11111`, `arn:aws-cn:ec2:cn-north-1:123456789012:instance/i-22222`, and `arn:aws-cn:ec2:cn-north-1:123456789012:instance/i-33333`) together into a custom component called `my-component`, for an application created for the resource group called `my-resource-group`, use the following command.

```
New-CWAIComponent -ResourceGroupName my-resource-group -ComponentName my-component -ResourceList arn:aws-cn:ec2:cn-north-1:123456789012:instance/i-11111,arn:aws-cn:ec2:cn-north-1:123456789012:instance/i-22222,arn:aws-cn:ec2:cn-north-1:123456789012:instance/i-33333
```

Ungroup a custom component

To ungroup a custom component using the Amazon CLI

To use the Amazon CLI to ungroup a custom component named `my-component` in an application created on the resource group, `my-resource-group`, use the following command.

```
aws application-insights delete-component --resource-group-name my-resource-group --component-name my-new-component
```

To ungroup a custom component using Amazon Tools for Windows PowerShell

To use the Amazon Tools for Windows PowerShell to ungroup a custom component named `my-component` in an application created on the resource group, `my-resource-group`, use the following command.

```
Remove-CWAIComponent -ComponentName my-component -ResourceGroupName my-resource-group
```

Update an application

Update an application using the Amazon CLI

You can use the Amazon CLI to update an application to generate Amazon Systems Manager OpsCenter OpsItems for problems detected with the application, and to associate the created OpsItems to the SNS topic `arn:aws-cn:sns:cn-north-1:123456789012:MyTopic`, using the following command.

```
aws application-insights update-application --resource-group-name my-resource-group --ops-center-enabled --ops-item-sns-topic-arn arn:aws-cn:sns:cn-north-1:123456789012:MyTopic
```

Update an application using Amazon Tools for Windows PowerShell

You can use the Amazon Tools for Windows PowerShell to update an application to generate Amazon SSM OpsCenter OpsItems for problems detected with the application, and to associate the created OpsItems to the SNS topic , using the following command.

```
Update-CWAIApplication -ResourceGroupName my-resource-group -OpsCenterEnabled true -OpsItemSNSTopicArn arn:aws-cn:sns:cn-north-1:123456789012:MyTopic
```

Update a custom component

Update a custom component using the Amazon CLI

You can use the Amazon CLI to update a custom component called `my-component` with a new component name, `my-new-component`, and an updated group of instances, by using the following command.

```
aws application-insights update-component --resource-group-name my-resource-group
 --component-name my-component --new-component-name my-new-component --resource-list arn:aws-cn:ec2:cn-north-1:123456789012:instance/i-44444 arn:aws-cn:ec2:cn-north-1:123456789012:instance/i-55555
```

Update a custom component using Amazon Tools for Windows PowerShell

You can use the Amazon Tools for Windows PowerShell to update a custom component called `my-component` with a new component name, `my-new-component`, and an updated group of instances, by using the following command.

```
Update-CWAIComponent -ComponentName my-component -NewComponentName my-new-component -ResourceGroupName my-resource-group -ResourceList arn:aws-cn:ec2:cn-north-1:123456789012:instance/i-44444,arn:aws-cn:ec2:cn-north-1:123456789012:instance/i-55555
```

Manage and update monitoring

You can manage and update monitoring for your Application Insights application using the command line.

Topics

- List problems with your application
- Describe an application problem
- Describe the anomalies or errors associated with a problem
- Describe an anomaly or error with the application
- Describe the monitoring configurations of a component
- Describe the recommended monitoring configuration of a component
- Update the monitoring configurations for a component
- Remove a specified resource group from Application Insights monitoring

List problems with your application

List problems with your application using the Amazon CLI

To use the Amazon CLI to list problems with your application detected between 1,000 and 10,000 milliseconds since Unix Epoch for an application created on a resource group called `my-resource-group`, use the following command.

```
aws application-insights list-problems --resource-group-name my-resource-group --start-time 1000 --end-time 10000
```

List problems with your application using Amazon Tools for Windows PowerShell

To use the Amazon Tools for Windows PowerShell to list problems with your application detected between 1,000 and 10,000 milliseconds since Unix Epoch for an application created on a resource group called `my-resource-group`, use the following command.

```
$startDate = "8/6/2019 3:33:00"
$endDate = "8/6/2019 3:34:00"
Get-CWAIProblemList -ResourceGroupName my-resource-group -StartTime $startDate -
EndTime $endDate
```

Describe an application problem

Describe an application problem using the Amazon CLI

To use the Amazon CLI to describe a problem with problem id `p-1234567890`, use the following command.

```
aws application-insights describe-problem —problem-id p-1234567890
```

Describe an application problem using Amazon Tools for Windows PowerShell

To use the Amazon Tools for Windows PowerShell to describe a problem with problem id `p-1234567890`, use the following command.

```
Get-CWAIProblem -ProblemId p-1234567890
```

Describe the anomalies or errors associated with a problem

Describe the anomalies or errors associated with a problem using the Amazon CLI

To use the Amazon CLI to describe the anomalies or errors associated with a problem with problem id `p-1234567890`, use the following command.

```
aws application-insights describe-problem-observations --problem-id -1234567890
```

Describe the anomalies or errors associated with a problem using Amazon Tools for Windows PowerShell

To use the Amazon Tools for Windows PowerShell to describe the anomalies or errors associated with a problem with problem id `p-1234567890`, use the following command.

```
Get-CWAIProblemObservation -ProblemId p-1234567890
```

Describe an anomaly or error with the application

Describe an anomaly or error with the application using the Amazon CLI

To use the Amazon CLI to describe an anomaly or error with the application with the observation id `o-1234567890`, use the following command.

```
aws application-insights describe-observation —observation-id o-1234567890
```

Describe an anomaly or error with the application using Amazon Tools for Windows PowerShell

To use the Amazon Tools for Windows PowerShell to describe an anomaly or error with the application with the observation id `o-1234567890`, use the following command.

```
Get-CWAIObservation -ObservationId o-1234567890
```

Describe the monitoring configurations of a component

Describe the monitoring configurations of a component using the Amazon CLI

To use the Amazon CLI to describe the monitoring configuration of a component called `my-component` in an application created on the resource group `my-resource-group`, use the following command.

```
aws application-insights describe-component-configuration —resource-group-name my-resource-group —component-name my-component
```

Describe the monitoring configurations of a component using Amazon Tools for Windows PowerShell

To use the Amazon Tools for Windows PowerShell to describe the monitoring configuration of a component called `my-component`, in an application created on the resource group `my-resource-group`, use the following command.

```
Get-CWAIComponentConfiguration -ComponentName my-component -ResourceGroupName my-resource-group
```

For more information about component configuration and for example JSON files, see Work with component configurations.

Describe the recommended monitoring configuration of a component

Describe the recommended monitoring configuration of a component using the Amazon CLI

When the component is part of a .NET Worker application, you can use the Amazon CLI to describe the recommended monitoring configuration of a component called `my-component` in an application created on the resource group `my-resource-group`, by using the following command.

```
aws application-insights describe-component-configuration-recommendation --resource-group-name my-resource-group --component-name my-component --tier DOT_NET_WORKER
```

Describe the recommended monitoring configuration of a component using Amazon Tools for Windows PowerShell

When the component is part of a .NET Worker application, you can use the Amazon Tools for Windows PowerShell to describe the recommended monitoring configuration of a component called `my-component` in an application created on the resource group `my-resource-group`, by using the following command.

```
Get-CWAIComponentConfigurationRecommendation -ComponentName my-component -ResourceGroupName my-resource-group -Tier DOT_NET_WORKER
```

For more information about component configuration and for example JSON files, see Work with component configurations.

Update the monitoring configurations for a component

Update the monitoring configurations for a component using the Amazon CLI

To use the Amazon CLI to update the component called `my-component` in an application created on the resource group called `my-resource-group`, use the following command. The command includes these actions:

1. Enable monitoring for the component.
2. Set the tier of the component to `.NET Worker`.
3. Update the JSON configuration of the component to read from the local file `configuration.txt`.

```
aws application-insights update-component-configuration --resource-group-name my-
resource-group --component-name my-component --tier DOT_NET_WORKER --monitor --
component-configuration "file://configuration.txt"
```

Update the monitoring configurations for a component using the Amazon Tools for Windows PowerShell

To use the Amazon Tools for Windows PowerShell to update the component called `my-component` in an application created on the resource group called `my-resource-group`, use the following command. The command includes these actions:

1. Enable monitoring for the component.
2. Set the tier of the component to `.NET Worker`.
3. Update the JSON configuration of the component to read from the local file `configuration.txt`.

```
[string]$config = Get-Content -Path configuration.txt
Update-CWAIComponentConfiguration -ComponentName my-component -ResourceGroupName my-
resource-group -Tier DOT_NET_WORKER -Monitor 1 -ComponentConfiguration $config
```

For more information about component configuration and for example JSON files, see Work with component configurations.

Remove a specified resource group from Application Insights monitoring

Remove a specified resource group from Application Insights monitoring using the Amazon CLI

To use the Amazon CLI to remove an application created on the resource group called `my-resource-group` from monitoring, use the following command.

```
aws application-insights delete-application --resource-group-name my-resource-group
```

Remove a specified resource group from Application Insights monitoring using the Amazon Tools for Windows PowerShell

To use the Amazon Tools for Windows PowerShell to remove an application created on the resource group called `my-resource-group` from monitoring, use the following command.

```
Remove-CWAIApplication -ResourceGroupName my-resource-group
```

Configure monitoring for SQL Always On Availability Groups

1. Create an application for the resource group with the SQL HA EC2 instances.

```
aws application-insights create-application --region <REGION> --resource-group-name
  <RESOURCE_GROUP_NAME>
```

2. Define the EC2 instances that represent the SQL HA cluster by creating a new application component.

```
aws application-insights create-component --resource-group-name
 "<RESOURCE_GROUP_NAME>" --component-name SQL_HA_CLUSTER --resource-list
 "arn:aws-cn:ec2:<REGION>:<ACCOUNT_ID>:instance/<CLUSTER_INSTANCE_1_ID>" "arn:aws-
cn:ec2:<REGION>:<ACCOUNT_ID>:instance/<CLUSTER_INSTANCE_2_ID>
```

3. Configure the SQL HA component.

```
aws application-insights  update-component-configuration --resource-group-name
 "<RESOURCE_GROUP_NAME>" --region <REGION> --component-name "SQL_HA_CLUSTER" --
monitor --tier SQL_SERVER_ALWAYSON_AVAILABILITY_GROUP --monitor  --component-
configuration '{
  "subComponents" : [ {
    "subComponentType" : "AWS::EC2::Instance",
    "alarmMetrics" : [ {
      "alarmMetricName" : "CPUUtilization",
      "monitor" : true
    }, {
      "alarmMetricName" : "StatusCheckFailed",
      "monitor" : true
    }, {
      "alarmMetricName" : "Processor % Processor Time",
      "monitor" : true
    }, {
      "alarmMetricName" : "Memory % Committed Bytes In Use",
      "monitor" : true
    }, {
      "alarmMetricName" : "Memory Available Mbytes",
      "monitor" : true
    }, {
      "alarmMetricName" : "Paging File % Usage",
```

```
      "monitor" : true
    }, {
      "alarmMetricName" : "System Processor Queue Length",
      "monitor" : true
    }, {
      "alarmMetricName" : "Network Interface Bytes Total/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "PhysicalDisk % Disk Time",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Buffer Manager Buffer cache hit ratio",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Buffer Manager Page life expectancy",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:General Statistics Processes blocked",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:General Statistics User Connections",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Locks Number of Deadlocks/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:SQL Statistics Batch Requests/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica File Bytes Received/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Log Bytes Received/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Log remaining for undo",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Log Send Queue",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Mirrored Write Transaction/
sec",
      "monitor" : true
```

```
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Recovery Queue",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Redo Bytes Remaining",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Redone Bytes/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Total Log requiring undo",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Transaction Delay",
      "monitor" : true
    } ],
    "windowsEvents" : [ {
      "logGroupName" : "WINDOWS_EVENTS-Application-<RESOURCE_GROUP_NAME>",
      "eventName" : "Application",
      "eventLevels" : [ "WARNING", "ERROR", "CRITICAL", "INFORMATION" ],
      "monitor" : true
    }, {
      "logGroupName" : "WINDOWS_EVENTS-System-<RESOURCE_GROUP_NAME>",
      "eventName" : "System",
      "eventLevels" : [ "WARNING", "ERROR", "CRITICAL" ],
      "monitor" : true
    }, {
      "logGroupName" : "WINDOWS_EVENTS-Security-<RESOURCE_GROUP_NAME>",
      "eventName" : "Security",
      "eventLevels" : [ "WARNING", "ERROR", "CRITICAL" ],
      "monitor" : true
    } ],
    "logs" : [ {
      "logGroupName" : "SQL_SERVER_ALWAYSON_AVAILABILITY_GROUP-
<RESOURCE_GROUP_NAME>",
      "logPath" : "C:\\Program Files\\Microsoft SQL Server\\MSSQL**.MSSQLSERVER\
\MSSQL\\Log\\ERRORLOG",
      "logType" : "SQL_SERVER",
      "monitor" : true,
      "encoding" : "utf-8"
    } ]
  }, {
    "subComponentType" : "AWS::EC2::Volume",
    "alarmMetrics" : [ {
```

```
        "alarmMetricName" : "VolumeReadBytes",
        "monitor" : true
      }, {
      "alarmMetricName" : "VolumeWriteBytes",
        "monitor" : true
      }, {
      "alarmMetricName" : "VolumeReadOps",
        "monitor" : true
      }, {
      "alarmMetricName" : "VolumeWriteOps",
        "monitor" : true
      }, {
      "alarmMetricName" : "VolumeQueueLength",
        "monitor" : true
      }, {
      "alarmMetricName" : "VolumeThroughputPercentage",
        "monitor" : true
      }, {
      "alarmMetricName" : "BurstBalance",
        "monitor" : true
      } ]
    } ]
}'
```

> **Note**
>
> Application Insights must ingest Application Event logs (information level) to detect cluster activities such as failover.

Configure monitoring for MySQL RDS

1. Create an application for the resource group with the RDS MySQL database instance.

```
aws application-insights create-application --region <REGION> --resource-group-name
  <RESOURCE_GROUP_NAME>
```

2. The error log is enabled by default. The slow query log can be enabled using data parameter groups. For more information, see Accessing the MySQL Slow Query and General Logs.

   • `set slow_query_log = 1`

- set log_output = FILE

3. Export the logs to be monitored to CloudWatch logs. For more information, see Publishing MySQL Logs to CloudWatch Logs.
4. Configure the MySQL RDS component.

```
aws application-insights  update-component-configuration --resource-group-name
 "<RESOURCE_GROUP_NAME>" --region <REGION> --component-name "<DB_COMPONENT_NAME>"
 --monitor --tier DEFAULT --monitor  --component-configuration "{\"alarmMetrics\":
[{\"alarmMetricName\":\"CPUUtilization\",\"monitor\":true}],\"logs\":[{\"logType\":
\"MYSQL\",\"monitor\":true},{\"logType\": \"MYSQL_SLOW_QUERY\",\"monitor\":false}]}"
```

Configure monitoring for MySQL EC2

1. Create an application for the resource group with the SQL HA EC2 instances.

```
aws application-insights create-application --region <REGION> --resource-group-name
  <RESOURCE_GROUP_NAME>
```

2. The error log is enabled by default. The slow query log can be enabled using data parameter groups. For more information, see Accessing the MySQL Slow Query and General Logs.
   - set slow_query_log = 1
   - set log_output = FILE
3. Configure the MySQL EC2 component.

```
aws application-insights  update-component-configuration --resource-group-name
 "<RESOURCE_GROUP_NAME>" --region <REGION> --component-name "<DB_COMPONENT_NAME>"
 --monitor --tier MYSQL --monitor  --component-configuration "{\"alarmMetrics\":
[{\"alarmMetricName\":\"CPUUtilization\",\"monitor\":true}],\"logs\":[{\"logGroupName
\":\"<UNIQUE_LOG_GROUP_NAME>\",\"logPath\":\"C:\\\\ProgramData\\\\MySQL\\\\MySQL
 Server **\\\\Data\\\\<FILE_NAME>.err\",\"logType\":\"MYSQL\",\"monitor\":true,
\"encoding\":\"utf-8\"}]}"
```

Configure monitoring for PostgreSQL RDS

1. Create an application for the resource group with the PostgreSQL RDS database instance.

```
aws application-insights create-application --region <REGION> --resource-group-name
  <RESOURCE_GROUP_NAME>
```

2. Publishing PostgreSQL logs to CloudWatch is not enabled by default. To enable monitoring, open the RDS console and select the database to monitor. Choose **Modify** in the upper right corner, and select the check box labeled **PostgreSQL** log. Choose **Continue** to save this setting.
3. Your PostgreSQL logs are exported to CloudWatch.
4. Configure the PostgreSQL RDS component.

```
aws application-insights update-component-configuration --region <REGION> --resource-
group-name <RESOURCE_GROUP_NAME> --component-name <DB_COMPONENT_NAME> --monitor --
tier DEFAULT --component-configuration
"{
   \"alarmMetrics\":[
      {
         \"alarmMetricName\": \"CPUUtilization\",
         \"monitor\": true
      }
   ],
   \"logs\":[
      {
         \"logType\": \"POSTGRESQL\",
         \"monitor\": true
      }
   ]
}"
```

Configure monitoring for PostgreSQL EC2

1. Create an application for the resource group with the PostgreSQL EC2 instance.

```
aws application-insights create-application --region <REGION> --resource-group-name
  <RESOURCE_GROUP_NAME>
```

2. Configure the PostgreSQL EC2 component.

```
aws application-insights update-component-configuration --region <REGION> --resource-
group-name <RESOURCE_GROUP_NAME> --component-name <DB_COMPONENT_NAME> --monitor --
tier POSTGRESQL --component-configuration
```

```
"{
   \"alarmMetrics\":[
      {
         \"alarmMetricName\":\"CPUUtilization\",
         \"monitor\":true
      }
   ],
   \"logs\":[
      {
         \"logGroupName\":\"<UNIQUE_LOG_GROUP_NAME>\",
         \"logPath\":\"/var/lib/pgsql/data/log/\",
         \"logType\":\"POSTGRESQL\",
         \"monitor\":true,
         \"encoding\":\"utf-8\"
      }
   ]
}"
```

Configure monitoring for Oracle RDS

1. Create an application for the resource group with the Oracle RDS database instance.

```
aws application-insights create-application --region <REGION> --resource-group-name
  <RESOURCE_GROUP_NAME>
```

2. Publishing Oracle logs to CloudWatch is not enabled by default. To enable monitoring, open the RDS console and select the database to monitor. Choose **Modify** in the upper right corner, and select the check boxes labeled **Alert** log and **Listener** log. Choose **Continue** to save this setting.
3. Your Oracle logs are exported to CloudWatch.
4. Configure the Oracle RDS component.

```
aws application-insights update-component-configuration --region <REGION> --resource-
group-name <RESOURCE_GROUP_NAME> --component-name <DB_COMPONENT_NAME> --monitor --
tier DEFAULT --component-configuration
"{
   \"alarmMetrics\":[
      {
         \"alarmMetricName\": \"CPUUtilization\",
         \"monitor\": true
      }
   ],
```

```
   \"logs\":[
      {
         \"logType\": \"ORACLE_ALERT\",
         \"monitor\": true
      },
      {
         \"logType\": \"ORACLE_LISTENER\",
         \"monitor\": true
      }
   ]
}"
```

Configure monitoring for Oracle EC2

1. Create an application for the resource group with the Oracle EC2 instance.

```
aws application-insights create-application --region <REGION> --resource-group-name
  <RESOURCE_GROUP_NAME>
```

2. Configure the Oracle EC2 component.

```
aws application-insights update-component-configuration --region <REGION> --resource-
group-name <RESOURCE_GROUP_NAME> --component-name <DB_COMPONENT_NAME> --monitor --
tier ORACLE --component-configuration
"{
   \"alarmMetrics\":[
      {
         \"alarmMetricName\":\"CPUUtilization\",
         \"monitor\":true
      }
   ],
   \"logs\":[
      {
         \"logGroupName\":\"<UNIQUE_LOG_GROUP_NAME>\",
         \"logPath\":\"/opt/oracle/diag/rdbms/*/*/trace\",
         \"logType\":\"ORACLE_ALERT\",
         \"monitor\":true,
      },
      {
         \"logGroupName\":\"<UNIQUE_LOG_GROUP_NAME>\",
         \"logPath\":\"/opt/oracle/diag/tnslsnr/$HOSTNAME/listener/trace/\",
         \"logType\":\"ORACLE_ALERT\",
```

```
            \"monitor\":true,
        }
    ]
}"
```

Application Insights CloudWatch Events for detected problems

For each application that is added to CloudWatch Application Insights, a CloudWatch event is published for the following events on a best effort basis:

- **Problem creation.** Emitted when CloudWatch Application Insights detects a new problem.
  - Detail Type: **"Application Insights Problem Detected"**
  - Detail:
    - `problemId`: The detected problem ID.
    - `region`: The Amazon Region where the problem was created.
    - `resourceGroupName`: The Resource Group for the registered application for which the problem was detected.
    - `status`: The status of the problem. Possible status and definitions are as follows:
      - `In progress`: A new problem has been identified. The problem is still receiving observations.
      - `Recovering`: The problem is stabilizing. You can manually resolve the problem when it is in this state.
      - `Resolved`: The problem is resolved. There are no new observations about this problem.
      - `Recurring`: The problem was resolved within the past 24 hours. It has reopened as a result of additional observations.
    - `severity`: The severity of the problem.
    - `problemUrl`: The console URL for the problem.

- **Problem update.** Emitted when the problem is updated with a new observation or when an existing observation is updated and the problem is subsequently updated; updates include a resolution or closure of the problem.
  - Detail Type: **"Application Insights Problem Updated"**
  - Detail:
    - `problemId`: The created problem ID.

- `region`: The Amazon Region where the problem was created.
- `resourceGroupName`: The Resource Group for the registered application for which the problem was detected.
- `status`: The status of the problem.
- `severity`: The severity of the problem.
- `problemUrl`: The console URL for the problem.

Receive notifications for detected problems

You can use Amazon SNS notifications, Systems Manager OpsCenter, or CloudWatch Events to receive notifications about problems that are detected in your applications.

CloudWatch Application Insights Amazon SNS notifications for detected problems

You can configure Amazon SNS notifications using the Amazon Web Services Management Console or the Amazon CLI. To set up notifications using the Amazon Web Services Management Console, you must have the necessary Amazon SNS permissions as shown in the following example.

```
{
    "Version": "2012-10-17",
    "Statement": [
      {
        "Effect": "Allow",
        "Action": [
          "sns:ListTopics",
          "sns:Subscribe",
          "sns:CreateTopic"
        ],
        "Resource": [
          "*"
        ]
      }
    ]
}
```

After you set up Amazon SNS notifications, you receive email notifications when a problem is created or resolved. You also receive notifications when a new observation is added to an existing problem.

The following example shows the content of an email notification.

```
    You are receiving this email because Problem "p-1234567" has been CREATED by Amazon
 CloudWatch Application Insights

    Problem Details:
    Problem URL: https:////console.aws.amazon.com/cloudwatch/home?region=us-
east-1#settings:AppInsightsSettings/problemDetails?problemId=p-1234567
    Problem Summary: Title of the problem
    Severity: HIGH
    Insights: Something specific is broken
    Status : RESOLVED
    AffectedResource: arn:aws:ec2:us-east-1:555555555555:host/testResource
    Region: us-east-1
    RecurringCount: 0
    StartTime: 2019-03-23T10:42:57.777Z
    LastUpdatedTime: 2019-03-23T21:49:37.777Z
    LastRecurrenceTime:
    StopTime: 2019-03-23T21:49:37.777Z

    Recent Issues
    - TelemetryArn:alarm1
      StartTime: 2024-08-15T22:12:46.007Z
      StopTime:
    - TelemetryArn:log-group1
      StartTime: 2024-08-15T22:12:46.007Z
      StopTime: 2024-08-15T22:12:46.007Z
```

How to receive problem notifications using Systems Manager

Actions through Amazon Systems Manager. CloudWatch Application Insights provides built-in integration with Systems Manager OpsCenter. If you choose to use this integration for your application, an OpsItem is created on the OpsCenter console for every problem detected with the application. From the OpsCenter console, you can view summarized information about the problem detected by CloudWatch Application Insights and pick a Systems Manager Automation runbook to take remedial actions or further identify Windows processes that are causing resource issues in your application.

How to receive problem notifications using CloudWatch Events

From the CloudWatch console, select **Rules** under **Events** in the left navigation pane. From the **Rules** page, select **Create rule**. Choose **Amazon CloudWatch Application Insights** from the

Service Name dropdown list and choose the **Event Type**. Then, choose **Add target** and select the target and parameters, for example, an **SNS topic** or **Lambda function**.

Application Insights cross-account observability

With CloudWatch Application Insights cross-account observability, you can monitor and troubleshoot your applications that span multiple Amazon accounts within a single Region.

You can use Amazon CloudWatch Observability Access Manager to set up one or more of your Amazon accounts as a monitoring account. You'll provide the monitoring account with the ability to view data in your source account by creating a sink in your monitoring account. You use the sink to create a link from your source account to your monitoring account. For more information, see CloudWatch cross-account observability.

Required resources

For proper functionality of CloudWatch Application Insights cross-account observability, ensure that the following telemetry types are shared through the CloudWatch Observability Access Manager.

- Applications in CloudWatch Application Insights
- Metrics in Amazon CloudWatch
- Log groups in Amazon CloudWatch Logs
- Traces in Amazon X-Ray

Work with component configurations

A component configuration is a text file in JSON format that describes the configuration settings of the component. This section provides an example template fragment, descriptions of component configuration sections, and example component configurations.

Topics

Component configuration template fragment

The following example shows a template fragment in JSON format.

```
{
  "alarmMetrics" : [
    list of alarm metrics
  ],
  "logs" : [
    list of logs
  ],
  "processes" : [
   list of processes
  ],
  "windowsEvents" : [
    list of windows events channels configurations
  ],
  "alarms" : [
    list of CloudWatch alarms
  ],
  "jmxPrometheusExporter": {
    JMX Prometheus Exporter configuration
  },
  "hanaPrometheusExporter": {
      SAP HANA Prometheus Exporter configuration
  },
  "haClusterPrometheusExporter": {
      HA Cluster Prometheus Exporter configuration
  },
  "netWeaverPrometheusExporter": {
      SAP NetWeaver Prometheus Exporter configuration
  },
  "subComponents" : [
    {
      "subComponentType" : "AWS::EC2::Instance" ...
      component nested instances configuration
    },
    {
      "subComponentType" : "AWS::EC2::Volume" ...
      component nested volumes configuration
    }
  ]
}
```

Component configuration sections

A component configuration includes several major sections. Sections in a component configuration can be listed in any order.

- **alarmMetrics (optional)**

  A list of metrics to monitor for the component. All component types can have an alarmMetrics section.
- **logs (optional)**

  A list of logs to monitor for the component. Only EC2 instances can have a logs section.
- **processes (optional)**

  A list of processes to monitor for the component. Only EC2 instances can have a processes section.
- **subComponents (optional)**

  Nested instance and volume subComponent configuration for the component. The following types of components can have nested instances and a subComponents section: ELB, ASG, custom-grouped EC2 instances , and EC2 instances.
- **alarms (optional)**

  A list of alarms to monitor for the component. All component types can have an alarm section.
- **windowsEvents (optional)**

  A list of windows events to monitor for the component. Only Windows on EC2 instances have a `windowsEvents` section.
- **JMXPrometheusExporter (optional)**

  JMXPrometheus Exporter configuration.
- **hanaPrometheusExporter (optional)**

  SAP HANA Prometheus Exporter configuration.
- **haClusterPrometheusExporter (optional)**

  HA Cluster Prometheus Exporter configuration.
- **netWeaverPrometheusExporter (optional)**

SAP NetWeaver Prometheus Exporter configuration.

- **sapAsePrometheusExporter (optional)**

  SAP ASE Prometheus Exporter configuration.

The following example shows the syntax for the **subComponents section fragment** in JSON format.

```
[
  {
    "subComponentType" : "AWS::EC2::Instance",
    "alarmMetrics" : [
      list of alarm metrics
    ],
    "logs" : [
      list of logs
    ],
    "processes": [
      list of processes
    ],
    "windowsEvents" : [
      list of windows events channels configurations
    ]
  },
  {
    "subComponentType" : "AWS::EC2::Volume",
    "alarmMetrics" : [
      list of alarm metrics
    ]
  }
]
```

Component configuration section properties

This section describes the properties of each component configuration section.

Sections

- Metric
- Log
- Process

- JMX Prometheus Exporter
- HANA Prometheus Exporter
- HA Cluster Prometheus Exporter
- NetWeaver Prometheus Exporter
- SAP ASE Prometheus Exporter
- Windows Events
- Alarm

Metric

Defines a metric to be monitored for the component.

JSON

```
{
  "alarmMetricName" : "monitoredMetricName",
  "monitor" : true/false
}
```

Properties

- **alarmMetricName (required)**

  The name of the metric to be monitored for the component. For metrics supported by Application Insights, see Logs and metrics supported by Amazon CloudWatch Application Insights.
- **monitor (optional)**

  Boolean to indicate whether to monitor the metric. The default value is `true`.

Log

Defines a log to be monitored for the component.

JSON

```
{
  "logGroupName" : "logGroupName",
  "logPath" : "logPath",
```

```
  "logType" : "logType",
  "encoding" : "encodingType",
  "monitor" : true/false
}
```

Properties

- **logGroupName (required)**

  The CloudWatch log group name to be associated to the monitored log. For the log group name constraints, see CreateLogGroup.

- **logPath (required for EC2 instance components; not required for components that do not use CloudWatch Agent, such as Amazon Lambda)**

  The path of the logs to be monitored. The log path must be an absolute Windows system file path. For more information, see CloudWatch Agent Configuration File: Logs Section.

- **logType (required)**

  The log type decides the log patterns against which Application Insights analyzes the log. The log type is selected from the following:

  - SQL_SERVER
  - MYSQL
  - MYSQL_SLOW_QUERY
  - POSTGRESQL
  - ORACLE_ALERT
  - ORACLE_LISTENER
  - IIS
  - APPLICATION
  - WINDOWS_EVENTS
  - WINDOWS_EVENTS_ACTIVE_DIRECTORY
  - WINDOWS_EVENTS_DNS
  - WINDOWS_EVENTS_IIS
  - WINDOWS_EVENTS_SHAREPOINT
  - SQL_SERVER_ALWAYSON_AVAILABILITY_GROUP
  - SQL_SERVER_FAILOVER_CLUSTER_INSTANCE

  - DEFAULT
  - CUSTOM
  - STEP_FUNCTION
  - API_GATEWAY_ACCESS
  - API_GATEWAY_EXECUTION
  - SAP_HANA_LOGS
  - SAP_HANA_TRACE
  - SAP_HANA_HIGH_AVAILABILITY
  - SAP_NETWEAVER_DEV_TRACE_LOGS
  - PACEMAKER_HIGH_AVAILABILITY
- **encoding (optional)**

  The type of encoding of the logs to be monitored. The specified encoding should be included in the list of CloudWatch agent supported encodings. If not provided, CloudWatch Application Insights uses the default encoding of type utf-8, except for:

  - SQL_SERVER: utf-16 encoding
  - IIS: ascii encoding
- **monitor (optional)**

  Boolean that indicates whether to monitor the logs. The default value is `true`.

Process

Defines a process to be monitored for the component.

JSON

```
{
  "processName" : "monitoredProcessName",
  "alarmMetrics" : [
      list of alarm metrics
  ]
}
```

Properties

- **processName (required)**

  The name of the process to be monitored for the component. The process name must not contain a process stem, such as `sqlservr` or `sqlservr.exe`.

- **alarmMetrics (required)**

  A list of metrics to monitor for this process. To view process metrics supported by CloudWatch Application Insights, see Amazon Elastic Compute Cloud (EC2).

JMX Prometheus Exporter

Defines the JMX Prometheus Exporter settings.

JSON

```
"JMXPrometheusExporter": {
  "jmxURL" : "JMX URL",
  "hostPort" : "The host and port",
  "prometheusPort" : "Target port to emit Prometheus metrics"
}
```

Properties

- **jmxURL (optional)**

  A complete JMX URL to connect to.

- **hostPort (optional)**

  The host and port to connect to through remote JMX. Only one of `jmxURL` and `hostPort` can be specified.

- **prometheusPort (optional)**

  The target port to send Prometheus metrics to. If not specified, the default port 9404 is used.

HANA Prometheus Exporter

Defines the HANA Prometheus Exporter settings.

JSON

```
"hanaPrometheusExporter": {
    "hanaSid": "SAP HANA  SID",
    "hanaPort": "HANA database port",
    "hanaSecretName": "HANA secret name",
    "prometheusPort": "Target port to emit Prometheus metrics"
}
```

Properties

- **hanaSid**

  The three-character SAP system ID (SID) of the SAP HANA system.

- **hanaPort**

  The HANA database port by which the exporter will query HANA metrics.

- **hanaSecretName**

  The Amazon Secrets Manager secret that stores HANA monitoring user credentials. The HANA Prometheus exporter uses these credentials to connect to the database and query HANA metrics.

- **prometheusPort (optional)**

  The target port to which Prometheus sends metrics. If not specified, the default port 9668 is used.

HA Cluster Prometheus Exporter

Defines the HA Cluster Prometheus Exporter settings.

JSON

```
"haClusterPrometheusExporter": {
    "prometheusPort": "Target port to emit Prometheus metrics"
}
```

Properties

- **prometheusPort (optional)**

  The target port to which Prometheus sends metrics. If not specified, the default port 9664 is used.

NetWeaver Prometheus Exporter

Defines the NetWeaver Prometheus Exporter settings.

JSON

```
"netWeaverPrometheusExporter": {
    "sapSid": "SAP NetWeaver  SID",
    "instanceNumbers": [ "Array of instance Numbers of SAP NetWeaver system "],
"prometheusPort": "Target port to emit Prometheus metrics"
}
```

Properties

- **sapSid**

  The 3 character SAP system ID (SID) of the SAP NetWeaver system.

- **instanceNumbers**

  Array of the instance Numbers of SAP NetWeaver system.

  Example: `"instanceNumbers": [ "00", "01"]`

- **prometheusPort (optional)**

  The target port to which to send Prometheus metrics. If not specified, the default port `9680` is used.

SAP ASE Prometheus Exporter

Defines the SAP ASE Prometheus Exporter settings.

JSON

```
"sapASEPrometheusExporter": {
    "sapAseSid": "SAP ASE SID",
    "sapAsePort": "SAP ASE database port",
    "sapAseSecretName": "SAP ASE secret name",
    "prometheusPort": "Target port to emit Prometheus metrics",
    "agreeToEnableASEMonitoring": true
}
```

Properties

- **sapAseSid**

  The three-character SAP system ID (SID) of the SAP ASE system.
- **sapAsePort**

  The SAP ASE database port by which the exporter will query ASE metrics.
- **sapAseSecretName**

  The Amazon Secrets Manager secret that stores ASE monitoring user credentials. The SAP ASE Prometheus exporter uses these credentials to connect to the database and query ASE metrics.
- **prometheusPort (optional)**

  The target port to which Prometheus sends metrics. If not specified, the default port 9399 is used. If there is another ASE DB that is using the default port, then we use 9499.

Windows Events

Defines Windows Events to log.

JSON

```
{
  "logGroupName" : "logGroupName",
  "eventName" : "eventName",
  "eventLevels" : ["ERROR","WARNING","CRITICAL","INFORMATION","VERBOSE"],
  "monitor" : true/false
}
```

Properties

- **logGroupName (required)**

  The CloudWatch log group name to be associated to the monitored log. For the log group name constraints, see CreateLogGroup.
- **eventName (required)**

The type of Windows Events to log. It is equivalent to the Windows Event log channel name. For example, System, Security, CustomEventName, etc. This field is required for each type of Windows event to log.

- **eventLevels (required)**

  The levels of event to log. You must specify each level to log. Possible values include `INFORMATION`, `WARNING`, `ERROR`, `CRITICAL`, and `VERBOSE`. This field is required for each type of Windows Event to log.

- **monitor (optional)**

  Boolean that indicates whether to monitor the logs. The default value is `true`.

Alarm

Defines a CloudWatch alarm to be monitored for the component.

JSON

```
{
  "alarmName" : "monitoredAlarmName",
  "severity" : HIGH/MEDIUM/LOW
}
```

Properties

- **alarmName (required)**

  The name of the CloudWatch alarm to be monitored for the component.

- **severity (optional)**

  Indicates the degree of outage when the alarm goes off.

Component configuration examples

The following examples show component configurations in JSON format for relevant services.

Example component configurations

- Amazon DynamoDB table

- Amazon EC2 Auto Scaling (ASG)
- Amazon EKS cluster
- Amazon Elastic Compute Cloud (EC2) instance
- Amazon Elastic Container Service (Amazon ECS)
- Amazon ECS services
- Amazon ECS tasks
- Amazon Elastic File System (Amazon EFS)
- Amazon FSx
- Amazon Relational Database Service (RDS) Aurora MySQL
- Amazon Relational Database Service (RDS) instance
- Amazon Route 53 health check
- Amazon Route 53 hosted zone
- Amazon Route 53 Resolver endpoint
- Amazon Route 53 Resolver query logging configuration
- Amazon S3 bucket
- Amazon Simple Queue Service (SQS)
- Amazon SNS topic
- Amazon Virtual Private Cloud (Amazon VPC)
- Amazon VPC Network Address Translation (NAT) gateways
- API Gateway REST API stages
- Application Elastic Load Balancing
- Amazon Lambda Function
- Amazon Network Firewall rule group
- Amazon Network Firewall rule group association
- Amazon Step Functions
- Customer-grouped Amazon EC2 instances
- Elastic Load Balancing
- Java
- Kubernetes on Amazon EC2
- RDS MariaDB and RDS MySQL

- RDS Oracle
- RDS PostgreSQL
- SAP ASE on Amazon EC2
- SAP ASE High Availability on Amazon EC2
- SAP HANA on Amazon EC2
- SAP HANA High Availability on Amazon EC2
- SAP NetWeaver on Amazon EC2
- SAP NetWeaver High Availability on Amazon EC2
- SQL Always On Availability Group
- SQL failover cluster instance

Amazon DynamoDB table

The following example shows a component configuration in JSON format for Amazon DynamoDB table.

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "SystemErrors",
      "monitor": false
    },
    {
      "alarmMetricName": "UserErrors",
      "monitor": false
    },
    {
      "alarmMetricName": "ConsumedReadCapacityUnits",
      "monitor": false
    },
    {
      "alarmMetricName": "ConsumedWriteCapacityUnits",
      "monitor": false
    },
    {
      "alarmMetricName": "ReadThrottleEvents",
      "monitor": false
    },
    {
```

```
      "alarmMetricName": "WriteThrottleEvents",
      "monitor": false
    },
    {
      "alarmMetricName": "ConditionalCheckFailedRequests",
      "monitor": false
    },
    {
      "alarmMetricName": "TransactionConflict",
      "monitor": false
    }
  ],
  "logs": []
}
```

Amazon EC2 Auto Scaling (ASG)

The following example shows a component configuration in JSON format for Amazon EC2 Auto Scaling (ASG).

```
{
    "alarmMetrics" : [
      {
        "alarmMetricName" : "CPUCreditBalance"
      }, {
        "alarmMetricName" : "EBSIOBalance%"
      }
    ],
    "subComponents" : [
      {
        "subComponentType" : "AWS::EC2::Instance",
        "alarmMetrics" : [
          {
            "alarmMetricName" : "CPUUtilization"
          }, {
            "alarmMetricName" : "StatusCheckFailed"
          }
        ],
        "logs" : [
          {
            "logGroupName" : "my_log_group",
            "logPath" : "C:\\LogFolder\\*",
            "logType" : "APPLICATION"
```

```
          }
        ],
        "processes" : [
          {
            "processName" : "my_process",
            "alarmMetrics" : [
              {
                  "alarmMetricName" : "procstat cpu_usage",
                  "monitor" : true
              }, {
                  "alarmMetricName" : "procstat memory_rss",
                  "monitor" : true
              }
          ]
      }
  ],
        "windowsEvents" : [
          {
            "logGroupName" : "my_log_group_2",
            "eventName" : "Application",
            "eventLevels" : [ "ERROR", "WARNING", "CRITICAL" ]
          }
        ]
      }, {
        "subComponentType" : "AWS::EC2::Volume",
        "alarmMetrics" : [
          {
            "alarmMetricName" : "VolumeQueueLength"
          }, {
            "alarmMetricName" : "BurstBalance"
          }
        ]
      }
    ],
    "alarms" : [
      {
        "alarmName" : "my_asg_alarm",
        "severity" : "LOW"
      }
    ]
  }
```

Amazon EKS cluster

The following example shows a component configuration in JSON format for Amazon EKS cluster.

```
{
    "alarmMetrics":[
       {
          "alarmMetricName": "cluster_failed_node_count",
          "monitor":true
       },
       {
          "alarmMetricName": "node_cpu_reserved_capacity",
          "monitor":true
       },
       {
          "alarmMetricName": "node_cpu_utilization",
          "monitor":true
       },
       {
          "alarmMetricName": "node_filesystem_utilization",
          "monitor":true
       },
       {
          "alarmMetricName": "node_memory_reserved_capacity",
          "monitor":true
       },
       {
          "alarmMetricName": "node_memory_utilization",
          "monitor":true
       },
       {
          "alarmMetricName": "node_network_total_bytes",
          "monitor":true
       },
       {
          "alarmMetricName": "pod_cpu_reserved_capacity",
          "monitor":true
       },
       {
          "alarmMetricName": "pod_cpu_utilization",
          "monitor":true
       },
       {
          "alarmMetricName": "pod_cpu_utilization_over_pod_limit",
```

```
            "monitor":true
        },
        {
            "alarmMetricName": "pod_memory_reserved_capacity",
            "monitor":true
        },
        {
            "alarmMetricName": "pod_memory_utilization",
            "monitor":true
        },
        {
            "alarmMetricName": "pod_memory_utilization_over_pod_limit",
            "monitor":true
        },
        {
            "alarmMetricName": "pod_network_rx_bytes",
            "monitor":true
        },
        {
            "alarmMetricName": "pod_network_tx_bytes",
            "monitor":true
        }
    ],
    "logs":[
        {
            "logGroupName": "/aws/containerinsights/kubernetes/application",
            "logType":"APPLICATION",
            "monitor":true,
            "encoding":"utf-8"
        }
    ],
    "subComponents":[
        {
            "subComponentType":"AWS::EC2::Instance",
            "alarmMetrics":[
                {
                    "alarmMetricName":"CPUUtilization",
                    "monitor":true
                },
                {
                    "alarmMetricName":"StatusCheckFailed",
                    "monitor":true
                },
                {
```

```
                "alarmMetricName":"disk_used_percent",
                "monitor":true
             },
             {
                "alarmMetricName":"mem_used_percent",
                "monitor":true
             }
          ],
          "logs":[
             {
                "logGroupName":"APPLICATION-KubernetesClusterOnEC2-IAD",
                "logPath":"",
                "logType":"APPLICATION",
                "monitor":true,
                "encoding":"utf-8"
             }
          ],
          "processes" : [
            {
                "processName" : "my_process",
                "alarmMetrics" : [
                    {
                        "alarmMetricName" : "procstat cpu_usage",
                        "monitor" : true
                    }, {
                        "alarmMetricName" : "procstat memory_rss",
                        "monitor" : true
                    }
                ]
            }
        ],
          "windowsEvents":[
             {
                "logGroupName":"my_log_group_2",
                "eventName":"Application",
                "eventLevels":[
                   "ERROR",
                   "WARNING",
                   "CRITICAL"
                ],
                "monitor":true
             }
          ]
       },
```

```
      {
         "subComponentType":"AWS::AutoScaling::AutoScalingGroup",
         "alarmMetrics":[
            {
               "alarmMetricName":"CPUCreditBalance",
               "monitor":true
            },
            {
               "alarmMetricName":"EBSIOBalance%",
               "monitor":true
            }
         ]
      },
      {
         "subComponentType":"AWS::EC2::Volume",
         "alarmMetrics":[
            {
               "alarmMetricName":"VolumeReadBytes",
               "monitor":true
            },
            {
               "alarmMetricName":"VolumeWriteBytes",
               "monitor":true
            },
            {
               "alarmMetricName":"VolumeReadOps",
               "monitor":true
            },
            {
               "alarmMetricName":"VolumeWriteOps",
               "monitor":true
            },
            {
               "alarmMetricName":"VolumeQueueLength",
               "monitor":true
            },
            {
               "alarmMetricName":"BurstBalance",
               "monitor":true
            }
         ]
      }
   ]
```

```
}
```

Note

- The `subComponents` section of `AWS::EC2::Instance`, `AWS::EC2::Volume`, and `AWS::AutoScaling::AutoScalingGroup` applies only to Amazon EKS cluster running on the EC2 launch type.
- The `windowsEvents` section of `AWS::EC2::Instance` in `subComponents` applies only to Windows running on Amazon EC2 instances.

Amazon Elastic Compute Cloud (EC2) instance

The following example shows a component configuration in JSON format for an Amazon EC2 instance.

Important

When an Amazon EC2 instance enters a `stopped` state, it is removed from monitoring. When it returns to a `running` state, it is added to the list of **Unmonitored components** on the **Application details** page of the CloudWatch Application Insights console. If automatic monitoring of new resources is enabled for the application, the instance is added to the list of **Monitored components**. However, the logs and metrics are set to the default for the workload. The previous log and metrics configuration is not saved.

```
{
    "alarmMetrics" : [
      {
        "alarmMetricName" : "CPUUtilization",
        "monitor" : true
      }, {
        "alarmMetricName" : "StatusCheckFailed"
      }
    ],
    "logs" : [
      {
        "logGroupName" : "my_log_group",
        "logPath" : "C:\\LogFolder\\*",
```

```
        "logType" : "APPLICATION",
        "monitor" : true
      },
      {
        "logGroupName" : "my_log_group_2",
        "logPath" : "C:\\LogFolder2\\*",
        "logType" : "IIS",
        "encoding" : "utf-8"
      }
    ],
    "processes" : [
        {
            "processName" : "my_process",
            "alarmMetrics" : [
                {
                    "alarmMetricName" : "procstat cpu_usage",
                    "monitor" : true
                }, {
                    "alarmMetricName" : "procstat memory_rss",
                    "monitor" : true
                }
            ]
        }
    ],
    "windowsEvents" : [
      {
        "logGroupName" : "my_log_group_3",
        "eventName" : "Application",
        "eventLevels" : [ "ERROR", "WARNING", "CRITICAL" ],
        "monitor" : true
      }, {
        "logGroupName" : "my_log_group_4",
        "eventName" : "System",
        "eventLevels" : [ "ERROR", "WARNING", "CRITICAL" ],
        "monitor" : true
    }],
     "alarms" : [
      {
        "alarmName" : "my_instance_alarm_1",
        "severity" : "HIGH"
      },
      {
        "alarmName" : "my_instance_alarm_2",
        "severity" : "LOW"
```

```
      }
    ],
     "subComponents" : [
     {
       "subComponentType" : "AWS::EC2::Volume",
       "alarmMetrics" : [
       {
         "alarmMetricName" : "VolumeQueueLength",
         "monitor" : "true"
       },
       {
         "alarmMetricName" : "VolumeThroughputPercentage",
         "monitor" : "true"
       },
       {
         "alarmMetricName" : "BurstBalance",
         "monitor" : "true"
       }]
    }]
  }
```

Amazon Elastic Container Service (Amazon ECS)

The following example shows a component configuration in JSON format for Amazon Elastic Container Service (Amazon ECS).

```
{
    "alarmMetrics":[
       {
          "alarmMetricName":"CpuUtilized",
          "monitor":true
       },
       {
          "alarmMetricName":"MemoryUtilized",
          "monitor":true
       },
       {
          "alarmMetricName":"NetworkRxBytes",
          "monitor":true
       },
       {
          "alarmMetricName":"NetworkTxBytes",
          "monitor":true
```

```
      },
      {
         "alarmMetricName":"RunningTaskCount",
         "monitor":true
      },
      {
         "alarmMetricName":"PendingTaskCount",
         "monitor":true
      },
      {
         "alarmMetricName":"StorageReadBytes",
         "monitor":true
      },
      {
         "alarmMetricName":"StorageWriteBytes",
         "monitor":true
      }
   ],
   "logs":[
      {
         "logGroupName":"/ecs/my-task-definition",
         "logType":"APPLICATION",
         "monitor":true
      }
   ],
   "subComponents":[
      {
         "subComponentType":"AWS::ElasticLoadBalancing::LoadBalancer",
         "alarmMetrics":[
            {
               "alarmMetricName":"HTTPCode_Backend_4XX",
               "monitor":true
            },
            {
               "alarmMetricName":"HTTPCode_Backend_5XX",
               "monitor":true
            },
            {
               "alarmMetricName":"Latency",
               "monitor":true
            },
            {
               "alarmMetricName":"SurgeQueueLength",
               "monitor":true
```

```
            },
            {
               "alarmMetricName":"UnHealthyHostCount",
               "monitor":true
            }
         ]
      },
      {
         "subComponentType":"AWS::ElasticLoadBalancingV2::LoadBalancer",
         "alarmMetrics":[
            {
               "alarmMetricName":"HTTPCode_Target_4XX_Count",
               "monitor":true
            },
            {
               "alarmMetricName":"HTTPCode_Target_5XX_Count",
               "monitor":true
            },
            {
               "alarmMetricName":"TargetResponseTime",
               "monitor":true
            },
            {
               "alarmMetricName":"UnHealthyHostCount",
               "monitor":true
            }
         ]
      },
      {
         "subComponentType":"AWS::EC2::Instance",
         "alarmMetrics":[
            {
               "alarmMetricName":"CPUUtilization",
               "monitor":true
            },
            {
               "alarmMetricName":"StatusCheckFailed",
               "monitor":true
            },
            {
               "alarmMetricName":"disk_used_percent",
               "monitor":true
            },
            {
```

```
                "alarmMetricName":"mem_used_percent",
                "monitor":true
            }
        ],
        "logs":[
            {
                "logGroupName":"my_log_group",
                "logPath":"/mylog/path",
                "logType":"APPLICATION",
                "monitor":true
            }
        ],
        "processes" : [
            {
                "processName" : "my_process",
                "alarmMetrics" : [
                    {
                        "alarmMetricName" : "procstat cpu_usage",
                        "monitor" : true
                    }, {
                        "alarmMetricName" : "procstat memory_rss",
                        "monitor" : true
                    }
                 ]
            }
        ],
        "windowsEvents":[
            {
                "logGroupName":"my_log_group_2",
                "eventName":"Application",
                "eventLevels":[
                    "ERROR",
                    "WARNING",
                    "CRITICAL"
                ],
                "monitor":true
            }
        ]
    },
    {
        "subComponentType":"AWS::EC2::Volume",
        "alarmMetrics":[
            {
                "alarmMetricName":"VolumeQueueLength",
```

```
                "monitor":"true"
             },
             {
                "alarmMetricName":"VolumeThroughputPercentage",
                "monitor":"true"
             },
             {
                "alarmMetricName":"BurstBalance",
                "monitor":"true"
             }
          ]
       }
    ]
 }
```

> **Note**
>
> - The `subComponents` section of `AWS::EC2::Instance` and `AWS::EC2::Volume` applies only to Amazon ECS clusters with ECS service or ECS task running on the EC2 launch type.
> - The `windowsEvents` section of `AWS::EC2::Instance` in `subComponents` applies only to Windows running on Amazon EC2 instances.

Amazon ECS services

The following example shows a component configuration in JSON format for an Amazon ECS service.

```
{
    "alarmMetrics":[
       {
          "alarmMetricName":"CPUUtilization",
          "monitor":true
       },
       {
          "alarmMetricName":"MemoryUtilization",
          "monitor":true
       },
       {
          "alarmMetricName":"CpuUtilized",
```

```
            "monitor":true
         },
         {
            "alarmMetricName":"MemoryUtilized",
            "monitor":true
         },
         {
            "alarmMetricName":"NetworkRxBytes",
            "monitor":true
         },
         {
            "alarmMetricName":"NetworkTxBytes",
            "monitor":true
         },
         {
            "alarmMetricName":"RunningTaskCount",
            "monitor":true
         },
         {
            "alarmMetricName":"PendingTaskCount",
            "monitor":true
         },
         {
            "alarmMetricName":"StorageReadBytes",
            "monitor":true
         },
         {
            "alarmMetricName":"StorageWriteBytes",
            "monitor":true
         }
      ],
      "logs":[
         {
            "logGroupName":"/ecs/my-task-definition",
            "logType":"APPLICATION",
            "monitor":true
         }
      ],
      "subComponents":[
         {
            "subComponentType":"AWS::ElasticLoadBalancing::LoadBalancer",
            "alarmMetrics":[
               {
                  "alarmMetricName":"HTTPCode_Backend_4XX",
```

```
                "monitor":true
            },
            {
                "alarmMetricName":"HTTPCode_Backend_5XX",
                "monitor":true
            },
            {
                "alarmMetricName":"Latency",
                "monitor":true
            },
            {
                "alarmMetricName":"SurgeQueueLength",
                "monitor":true
            },
            {
                "alarmMetricName":"UnHealthyHostCount",
                "monitor":true
            }
        ]
    },
    {
        "subComponentType":"AWS::ElasticLoadBalancingV2::LoadBalancer",
        "alarmMetrics":[
            {
                "alarmMetricName":"HTTPCode_Target_4XX_Count",
                "monitor":true
            },
            {
                "alarmMetricName":"HTTPCode_Target_5XX_Count",
                "monitor":true
            },
            {
                "alarmMetricName":"TargetResponseTime",
                "monitor":true
            },
            {
                "alarmMetricName":"UnHealthyHostCount",
                "monitor":true
            }
        ]
    },
    {
        "subComponentType":"AWS::EC2::Instance",
        "alarmMetrics":[
```

```
            {
               "alarmMetricName":"CPUUtilization",
               "monitor":true
            },
            {
               "alarmMetricName":"StatusCheckFailed",
               "monitor":true
            },
            {
               "alarmMetricName":"disk_used_percent",
               "monitor":true
            },
            {
               "alarmMetricName":"mem_used_percent",
               "monitor":true
            }
         ],
         "logs":[
            {
               "logGroupName":"my_log_group",
               "logPath":"/mylog/path",
               "logType":"APPLICATION",
               "monitor":true
            }
         ],
         "processes" : [
            {
               "processName" : "my_process",
               "alarmMetrics" : [
               {
                   "alarmMetricName" : "procstat cpu_usage",
                   "monitor" : true
               }, {
                   "alarmMetricName" : "procstat memory_rss",
                   "monitor" : true
               }
           ]
       }
   ],
         "windowsEvents":[
            {
               "logGroupName":"my_log_group_2",
               "eventName":"Application",
               "eventLevels":[
```

```
                    "ERROR",
                    "WARNING",
                    "CRITICAL"
                 ],
                 "monitor":true
              }
           ]
        },
        {
           "subComponentType":"AWS::EC2::Volume",
           "alarmMetrics":[
              {
                 "alarmMetricName":"VolumeQueueLength",
                 "monitor":"true"
              },
              {
                 "alarmMetricName":"VolumeThroughputPercentage",
                 "monitor":"true"
              },
              {
                 "alarmMetricName":"BurstBalance",
                 "monitor":"true"
              }
           ]
        }
     ]
  }
```

Note

- The `subComponents` section of `AWS::EC2::Instance` and `AWS::EC2::Volume` applies only to Amazon ECS running on the EC2 launch type.
- The `windowsEvents` section of `AWS::EC2::Instance` in `subComponents` applies only to Windows running on Amazon EC2 instances.

Amazon ECS tasks

The following example shows a component configuration in JSON format for an Amazon ECS task.

```
{
```

```
    "logs":[
       {
          "logGroupName":"/ecs/my-task-definition",
          "logType":"APPLICATION",
          "monitor":true
       }
    ],
    "processes" : [
        {
            "processName" : "my_process",
            "alarmMetrics" : [
                {
                    "alarmMetricName" : "procstat cpu_usage",
                    "monitor" : true
                }, {
                    "alarmMetricName" : "procstat memory_rss",
                    "monitor" : true
                }
            ]
        }
    ]
 }
```

Amazon Elastic File System (Amazon EFS)

The following example shows a component configuration in JSON format for Amazon EFS.

```
{
   "alarmMetrics": [
     {
       "alarmMetricName": "BurstCreditBalance",
       "monitor": true
     },
     {
       "alarmMetricName": "PercentIOLimit",
       "monitor": true
     },
     {
       "alarmMetricName": "PermittedThroughput",
       "monitor": true
     },
     {
       "alarmMetricName": "MeteredIOBytes",
       "monitor": true
```

```
    },
    {
      "alarmMetricName": "TotalIOBytes",
      "monitor": true
    },
    {
      "alarmMetricName": "DataWriteIOBytes",
      "monitor": true
    },
    {
      "alarmMetricName": "DataReadIOBytes",
      "monitor": true
    },
    {
      "alarmMetricName": "MetadataIOBytes",
      "monitor": true
    },
    {
      "alarmMetricName": "ClientConnections",
      "monitor": true
    },
    {
      "alarmMetricName": "TimeSinceLastSync",
      "monitor": true
    },
    {
      "alarmMetricName": "Throughput",
      "monitor": true
    },
    {
      "alarmMetricName": "PercentageOfPermittedThroughputUtilization",
      "monitor": true
    },
    {
      "alarmMetricName": "ThroughputIOPS",
      "monitor": true
    },
    {
      "alarmMetricName": "PercentThroughputDataReadIOBytes",
      "monitor": true
    },
    {
      "alarmMetricName": "PercentThroughputDataWriteIOBytes",
      "monitor": true
```

```
      },
      {
        "alarmMetricName": "PercentageOfIOPSDataReadIOBytes",
        "monitor": true
      },
      {
        "alarmMetricName": "PercentageOfIOPSDataWriteIOBytes",
        "monitor": true
      },
      {
        "alarmMetricName": "AverageDataReadIOBytesSize",
        "monitor": true
      },
      {
        "alarmMetricName": "AverageDataWriteIOBytesSize",
        "monitor": true
      }
    ],
    "logs": [
     {
     "logGroupName": "/aws/efs/utils",
     "logType": "EFS_MOUNT_STATUS",
     "monitor": true,
     }
    ]
  }
```

Amazon FSx

The following example shows a component configuration in JSON format for Amazon FSx.

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "DataReadBytes",
      "monitor": true
    },
    {
      "alarmMetricName": "DataWriteBytes",
      "monitor": true
    },
    {
      "alarmMetricName": "DataReadOperations",
      "monitor": true
```

```
    },
    {
      "alarmMetricName": "DataWriteOperations",
      "monitor": true
    },
    {
      "alarmMetricName": "MetadataOperations",
      "monitor": true
    },
    {
      "alarmMetricName": "FreeStorageCapacity",
      "monitor": true
    }
  ]
}
```

Amazon Relational Database Service (RDS) Aurora MySQL

The following example shows a component configuration in JSON format for Amazon RDS Aurora MySQL.

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "CPUUtilization",
      "monitor": true
    },
    {
      "alarmMetricName": "CommitLatency",
      "monitor": true
    }
  ],
  "logs": [
    {
      "logType": "MYSQL",
      "monitor": true,
    },
    {
      "logType": "MYSQL_SLOW_QUERY",
      "monitor": false
    }
  ]
}
```

Amazon Relational Database Service (RDS) instance

The following example shows a component configuration in JSON format for an Amazon RDS instance.

```
{
  "alarmMetrics" : [
    {
      "alarmMetricName" : "BurstBalance",
      "monitor" : true
    }, {
      "alarmMetricName" : "WriteThroughput",
      "monitor" : false
    }
  ],

  "alarms" : [
    {
      "alarmName" : "my_rds_instance_alarm",
      "severity" : "MEDIUM"
    }
  ]
}
```

Amazon Route 53 health check

The following example shows a component configuration in JSON format for Amazon Route 53 health check.

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "ChildHealthCheckHealthyCount",
      "monitor": true
    },
    {
      "alarmMetricName": "ConnectionTime",
      "monitor": true
    },
    {
      "alarmMetricName": "HealthCheckPercentageHealthy",
      "monitor": true
    },
```

```
    {
      "alarmMetricName": "HealthCheckStatus",
      "monitor": true
    },
    {
      "alarmMetricName": "SSLHandshakeTime",
      "monitor": true
    },
    {
      "alarmMetricName": "TimeToFirstByte",
      "monitor": true
    }
  ]
}
```

Amazon Route 53 hosted zone

The following example shows a component configuration in JSON format for Amazon Route 53 hosted zone.

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "DNSQueries",
      "monitor": true
    },
    {
      "alarmMetricName": "DNSSECInternalFailure",
      "monitor": true
    },
    {
      "alarmMetricName": "DNSSECKeySigningKeysNeedingAction",
      "monitor": true
    },
    {
      "alarmMetricName": "DNSSECKeySigningKeyMaxNeedingActionAge",
      "monitor": true
    },
    {
      "alarmMetricName": "DNSSECKeySigningKeyAge",
      "monitor": true
    }
  ],
```

```
  "logs": [
    {
      "logGroupName":"/hosted-zone/logs",
      "logType": "ROUTE53_DNS_PUBLIC_QUERY_LOGS",
      "monitor": true
    }
  ]
}
```

Amazon Route 53 Resolver endpoint

The following example shows a component configuration in JSON format for Amazon Route 53 Resolver endpoint.

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "EndpointHealthyENICount",
      "monitor": true
    },
    {
      "alarmMetricName": "EndpointUnHealthyENICount",
      "monitor": true
    },
    {
      "alarmMetricName": "InboundQueryVolume",
      "monitor": true
    },
    {
      "alarmMetricName": "OutboundQueryVolume",
      "monitor": true
    },
    {
      "alarmMetricName": "OutboundQueryAggregateVolume",
      "monitor": true
    }
  ]
}
```

Amazon Route 53 Resolver query logging configuration

The following example shows a component configuration in JSON format for Amazon Route 53 Resolver query logging configuration.

```
{
  "logs": [
    {
      "logGroupName": "/resolver-query-log-config/logs",
      "logType": "ROUTE53_RESOLVER_QUERY_LOGS",
      "monitor": true
    }
  ]
}
```

Amazon S3 bucket

The following example shows a component configurations in JSON format for Amazon S3 bucket.

```
{
    "alarmMetrics" : [
        {
            "alarmMetricName" : "ReplicationLatency",
            "monitor" : true
        }, {
            "alarmMetricName" : "5xxErrors",
            "monitor" : true
        }, {
            "alarmMetricName" : "BytesDownloaded"
            "monitor" : true
        }
    ]
}
```

Amazon Simple Queue Service (SQS)

The following example shows a component configuration in JSON format for Amazon Simple Queue Service.

```
{
  "alarmMetrics" : [
    {
      "alarmMetricName" : "ApproximateAgeOfOldestMessage"
    }, {
      "alarmMetricName" : "NumberOfEmptyReceives"
    }
  ],
```

```
  "alarms" : [
    {
      "alarmName" : "my_sqs_alarm",
      "severity" : "MEDIUM"
    }
  ]
}
```

Amazon SNS topic

The following example shows a component configuration in JSON format for Amazon SNS topic.

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "NumberOfNotificationsFailed",
      "monitor": true
    },
    {
      "alarmMetricName": "NumberOfNotificationsFilteredOut-InvalidAttributes",
      "monitor": true
    },
    {
      "alarmMetricName": "NumberOfNotificationsFilteredOut-NoMessageAttributes",
      "monitor": true
    },
    {
      "alarmMetricName": "NumberOfNotificationsFailedToRedriveToDlq",
      "monitor": true
    }
  ]
}
```

Amazon Virtual Private Cloud (Amazon VPC)

The following example shows a component configuration in JSON format for Amazon VPC.

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "NetworkAddressUsage",
      "monitor": true
    },
```

```
    {
      "alarmMetricName": "NetworkAddressUsagePeered",
      "monitor": true
    },
    {
      "alarmMetricName": "VPCFirewallQueryVolume",
      "monitor": true
    }
  ]
}
```

Amazon VPC Network Address Translation (NAT) gateways

The following example shows a component configuration in JSON format for NAT gateways.

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "ErrorPortAllocation",
      "monitor": true
    },
    {
      "alarmMetricName": "IdleTimeoutCount",
      "monitor": true
    }
  ]
}
```

API Gateway REST API stages

The following example shows a component configuration in JSON format for API Gateway REST API stages.

```
{
     "alarmMetrics" : [
         {
             "alarmMetricName" : "4XXError",
             "monitor" : true
         },
         {
             "alarmMetricName" : "5XXError",
             "monitor" : true
         }
```

```
    ],
    "logs" : [
        {
            "logType" : "API_GATEWAY_EXECUTION",
            "monitor" : true
        },
        {
            "logType" : "API_GATEWAY_ACCESS",
            "monitor" : true
        }
    ]
}
```

Application Elastic Load Balancing

The following example shows a component configuration in JSON format for Application Elastic Load Balancing.

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "ActiveConnectionCount",
    }, {
      "alarmMetricName": "TargetResponseTime"
    }
  ],
  "subComponents": [
    {
      "subComponentType": "AWS::EC2::Instance",
      "alarmMetrics": [
        {
          "alarmMetricName": "CPUUtilization",
        }, {
          "alarmMetricName": "StatusCheckFailed"
        }
      ],
      "logs": [
        {
          "logGroupName": "my_log_group",
          "logPath": "C:\\LogFolder\\*",
          "logType": "APPLICATION",
        }
      ],
```

```
      "windowsEvents": [
        {
          "logGroupName": "my_log_group_2",
          "eventName": "Application",
          "eventLevels": [ "ERROR", "WARNING", "CRITICAL" ]
        }
      ]
    }, {
      "subComponentType": "AWS::EC2::Volume",
      "alarmMetrics": [
        {
          "alarmMetricName": "VolumeQueueLength",
        }, {
          "alarmMetricName": "BurstBalance"
        }
      ]
    }
  ],

  "alarms": [
    {
      "alarmName": "my_alb_alarm",
      "severity": "LOW"
    }
  ]
}
```

Amazon Lambda Function

The following example shows a component configuration in JSON format for Amazon Lambda Function.

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "Errors",
      "monitor": true
    },
    {
      "alarmMetricName": "Throttles",
      "monitor": true
    },
    {
```

```
      "alarmMetricName": "IteratorAge",
      "monitor": true
    },
    {
      "alarmMetricName": "Duration",
      "monitor": true
    }
  ],
  "logs": [
    {
      "logType": "DEFAULT",
      "monitor": true
    }
  ]
}
```

Amazon Network Firewall rule group

The following example shows a component configuration in JSON format for Amazon Network Firewall rule group.

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "FirewallRuleGroupQueryVolume",
      "monitor": true
    }
  ]
}
```

Amazon Network Firewall rule group association

The following example shows a component configuration in JSON format for Amazon Network Firewall rule group association.

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "FirewallRuleGroupQueryVolume",
      "monitor": true
    }
  ]
```

```
}
```

Amazon Step Functions

The following example shows a component configurations in JSON format for Amazon Step Functions.

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "ExecutionsFailed",
      "monitor": true
    },
    {
      "alarmMetricName": "LambdaFunctionsFailed",
      "monitor": true
    },
    {
      "alarmMetricName": "ProvisionedRefillRate",
      "monitor": true
    }
  ],
  "logs": [
    {
     "logGroupName": "/aws/states/HelloWorld-Logs",
      "logType": "STEP_FUNCTION",
      "monitor": true,
    }
  ]
}
```

Customer-grouped Amazon EC2 instances

The following example shows a component configuration in JSON format for customer-grouped Amazon EC2 instances.

```
{
    "subComponents": [
      {
        "subComponentType": "AWS::EC2::Instance",
        "alarmMetrics": [
          {
            "alarmMetricName": "CPUUtilization",
```

```
        },
        {
          "alarmMetricName": "StatusCheckFailed"
        }
      ],
      "logs": [
        {
          "logGroupName": "my_log_group",
          "logPath": "C:\\LogFolder\\*",
          "logType": "APPLICATION",
        }
      ],
      "processes": [
          {
              "processName": "my_process",
              "alarmMetrics": [
                  {
                      "alarmMetricName": "procstat cpu_usage",
                      "monitor": true
                  }, {
                      "alarmMetricName": "procstat memory_rss",
                      "monitor": true
                  }
              ]
          }
      ],
      "windowsEvents": [
        {
          "logGroupName": "my_log_group_2",
          "eventName": "Application",
          "eventLevels": [ "ERROR", "WARNING", "CRITICAL" ]
        }
      ]
    }, {
      "subComponentType": "AWS::EC2::Volume",
      "alarmMetrics": [
        {
          "alarmMetricName": "VolumeQueueLength",
        }, {
          "alarmMetricName": "BurstBalance"
        }
      ]
    }
  ],
```

```
    "alarms": [
      {
        "alarmName": "my_alarm",
        "severity": "MEDIUM"
      }
    ]
  }
```

Elastic Load Balancing

The following example shows a component configuration in JSON format for Elastic Load Balancing.

```
{
    "alarmMetrics": [
      {
        "alarmMetricName": "EstimatedALBActiveConnectionCount"
      }, {
        "alarmMetricName": "HTTPCode_Backend_5XX"
      }
    ],
    "subComponents": [
      {
        "subComponentType": "AWS::EC2::Instance",
        "alarmMetrics": [
          {
            "alarmMetricName": "CPUUtilization"
          }, {
            "alarmMetricName": "StatusCheckFailed"
          }
        ],
        "logs": [
          {
            "logGroupName": "my_log_group",
            "logPath": "C:\\LogFolder\\*",
            "logType": "APPLICATION"
          }
        ],
        "processes": [
          {
            "processName": "my_process",
            "alarmMetrics": [
                {
```

```
                "alarmMetricName": "procstat cpu_usage",
                "monitor": true
              }, {
                "alarmMetricName": "procstat memory_rss",
                "monitor": true
              }
            ]
        }
      ],
      "windowsEvents": [
        {
          "logGroupName": "my_log_group_2",
          "eventName": "Application",
          "eventLevels": [ "ERROR", "WARNING", "CRITICAL" ],
          "monitor": true
        }
      ]
    }, {
      "subComponentType": "AWS::EC2::Volume",
      "alarmMetrics": [
        {
          "alarmMetricName": "VolumeQueueLength"
        }, {
          "alarmMetricName": "BurstBalance"
        }
      ]
    }
  ],

  "alarms": [
    {
      "alarmName": "my_elb_alarm",
      "severity": "HIGH"
    }
  ]
}
```

Java

The following example shows a component configuration in JSON format for Java.

```
{
  "alarmMetrics": [ {
    "alarmMetricName": "java_lang_threading_threadcount",
```

```
      "monitor": true
    },
    {
      "alarmMetricName": "java_lang_memory_heapmemoryusage_used",
      "monitor": true
    },
    {
      "alarmMetricName": "java_lang_memory_heapmemoryusage_committed",
      "monitor": true
    }],
    "logs": [ ],
    "JMXPrometheusExporter": {
        "hostPort": "8686",
        "prometheusPort": "9404"
    }
}
```

Note

Application Insights does not support configuring authentication for Prometheus JMX exporter. For information about how to set up authentication, see the Prometheus JMX exporter example configuration.

Kubernetes on Amazon EC2

The following example shows a component configuration in JSON format for Kubernetes on Amazon EC2.

```
{
    "alarmMetrics":[
       {
          "alarmMetricName":"cluster_failed_node_count",
          "monitor":true
       },
       {
          "alarmMetricName":"node_cpu_reserved_capacity",
          "monitor":true
       },
       {
          "alarmMetricName":"node_cpu_utilization",
          "monitor":true
```

```
        },
        {
           "alarmMetricName":"node_filesystem_utilization",
           "monitor":true
        },
        {
           "alarmMetricName":"node_memory_reserved_capacity",
           "monitor":true
        },
        {
           "alarmMetricName":"node_memory_utilization",
           "monitor":true
        },
        {
           "alarmMetricName":"node_network_total_bytes",
           "monitor":true
        },
        {
           "alarmMetricName":"pod_cpu_reserved_capacity",
           "monitor":true
        },
        {
           "alarmMetricName":"pod_cpu_utilization",
           "monitor":true
        },
        {
           "alarmMetricName":"pod_cpu_utilization_over_pod_limit",
           "monitor":true
        },
        {
           "alarmMetricName":"pod_memory_reserved_capacity",
           "monitor":true
        },
        {
           "alarmMetricName":"pod_memory_utilization",
           "monitor":true
        },
        {
           "alarmMetricName":"pod_memory_utilization_over_pod_limit",
           "monitor":true
        },
        {
           "alarmMetricName":"pod_network_rx_bytes",
           "monitor":true
```

```
        },
        {
           "alarmMetricName":"pod_network_tx_bytes",
           "monitor":true
        }
     ],
     "logs":[
        {
           "logGroupName":"/aws/containerinsights/kubernetes/application",
           "logType":"APPLICATION",
           "monitor":true,
           "encoding":"utf-8"
        }
     ],
     "subComponents":[
        {
           "subComponentType":"AWS::EC2::Instance",
           "alarmMetrics":[
              {
                 "alarmMetricName":"CPUUtilization",
                 "monitor":true
              },
              {
                 "alarmMetricName":"StatusCheckFailed",
                 "monitor":true
              },
              {
                 "alarmMetricName":"disk_used_percent",
                 "monitor":true
              },
              {
                 "alarmMetricName":"mem_used_percent",
                 "monitor":true
              }
           ],
           "logs":[
              {
                 "logGroupName":"APPLICATION-KubernetesClusterOnEC2-IAD",
                 "logPath":"",
                 "logType":"APPLICATION",
                 "monitor":true,
                 "encoding":"utf-8"
              }
           ],
```

```
        "processes" : [
          {
              "processName" : "my_process",
              "alarmMetrics" : [
                  {
                      "alarmMetricName" : "procstat cpu_usage",
                      "monitor" : true
                  }, {
                      "alarmMetricName" : "procstat memory_rss",
                      "monitor" : true
                  }
              ]
          }
      ]
     },
     {
        "subComponentType":"AWS::EC2::Volume",
        "alarmMetrics":[
           {
              "alarmMetricName":"VolumeReadBytes",
              "monitor":true
           },
           {
              "alarmMetricName":"VolumeWriteBytes",
              "monitor":true
           },
           {
              "alarmMetricName":"VolumeReadOps",
              "monitor":true
           },
           {
              "alarmMetricName":"VolumeWriteOps",
              "monitor":true
           },
           {
              "alarmMetricName":"VolumeQueueLength",
              "monitor":true
           },
           {
              "alarmMetricName":"BurstBalance",
              "monitor":true
           }
        ]
     }
```

```
    ]
 }
```

RDS MariaDB and RDS MySQL

The following example shows a component configuration in JSON format for RDS MariaDB and RDS MySQL.

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "CPUUtilization",
      "monitor": true
    }
  ],
  "logs": [
    {
      "logType": "MYSQL",
      "monitor": true,
    },
    {
      "logType": "MYSQL_SLOW_QUERY",
      "monitor": false
    }
  ]
}
```

RDS Oracle

The following example shows a component configuration in JSON format for RDS Oracle.

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "CPUUtilization",
      "monitor": true
    }
  ],
  "logs": [
    {
      "logType": "ORACLE_ALERT",
      "monitor": true,
    },
```

```
    {
      "logType": "ORACLE_LISTENER",
      "monitor": false
    }
  ]
}
```

RDS PostgreSQL

The following example shows a component configurations in JSON format for RDS PostgreSQL.

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "CPUUtilization",
      "monitor": true
    }
  ],
  "logs": [
    {
      "logType": "POSTGRESQL",
      "monitor": true
    }
  ]
}
```

SAP ASE on Amazon EC2

The following example shows a component configuration in JSON format for SAP ASE on Amazon EC2.

```
{
  "subComponents": [
    {
      "subComponentType": "AWS::EC2::Instance",
      "alarmMetrics": [
        {
          "alarmMetricName": "asedb_database_availability",
          "monitor": true
        },
        {
          "alarmMetricName": "asedb_trunc_log_on_chkpt_enabled",
          "monitor": true
```

```
        },
        {
          "alarmMetricName": "asedb_last_db_backup_age_in_days",
          "monitor": true
        },
        {
          "alarmMetricName": "asedb_last_transaction_log_backup_age_in_hours",
          "monitor": true
        },
        {
          "alarmMetricName": "asedb_suspected_database",
          "monitor": true
        },
        {
          "alarmMetricName": "asedb_db_space_usage_percent",
          "monitor": true
        },
        {
          "alarmMetricName": "asedb_db_log_space_usage_percent",
          "monitor": true
        },
        {
          "alarmMetricName": "asedb_locked_login",
          "monitor": true
        },
        {
          "alarmMetricName": "asedb_data_cache_hit_ratio",
          "monitor": true
        }
      ],
      "logs": [
        {
          "logGroupName": "SAP_ASE_SERVER_LOGS-my-resource-group",
          "logPath": "/sybase/SY2/ASE-*/install/SY2.log",
          "logType": "SAP_ASE_SERVER_LOGS",
          "monitor": true,
          "encoding": "utf-8"
        },
        {
          "logGroupName": "SAP_ASE_BACKUP_SERVER_LOGS-my-resource-group",
          "logPath": "/sybase/SY2/ASE-*/install/SY2_BS.log",
          "logType": "SAP_ASE_BACKUP_SERVER_LOGS",
          "monitor": true,
          "encoding": "utf-8"
```

```
        }
  ],
  "sapAsePrometheusExporter": {
    "sapAseSid": "ASE",
    "sapAsePort": "4901",
    "sapAseSecretName": "ASE_DB_CREDS",
    "prometheusPort": "9399",
    "agreeToEnableASEMonitoring": true
}
```

SAP ASE High Availability on Amazon EC2

The following example shows a component configuration in JSON format for SAP ASE High Availability on Amazon EC2.

```
{
  "subComponents": [
    {
      "subComponentType": "AWS::EC2::Instance",
      "alarmMetrics": [
        {
          "alarmMetricName": "asedb_database_availability",
          "monitor": true
        },
        {
          "alarmMetricName": "asedb_trunc_log_on_chkpt_enabled",
          "monitor": true
        },
        {
          "alarmMetricName": "asedb_last_db_backup_age_in_days",
          "monitor": true
        },
        {
          "alarmMetricName": "asedb_last_transaction_log_backup_age_in_hours",
          "monitor": true
        },
        {
          "alarmMetricName": "asedb_suspected_database",
          "monitor": true
        },
        {
          "alarmMetricName": "asedb_db_space_usage_percent",
          "monitor": true
```

```
      },
      {
        "alarmMetricName": "asedb_ha_replication_state",
        "monitor": true
      },
      {
        "alarmMetricName": "asedb_ha_replication_mode",
        "monitor": true
      },
      {
        "alarmMetricName": "asedb_ha_replication_latency_in_minutes",
        "monitor": true
      }
    ],
    "logs": [
      {
        "logGroupName": "SAP_ASE_SERVER_LOGS-my-resource-group",
        "logPath": "/sybase/SY2/ASE-*/install/SY2.log",
        "logType": "SAP_ASE_SERVER_LOGS",
        "monitor": true,
        "encoding": "utf-8"
      },
      {
        "logGroupName": "SAP_ASE_BACKUP_SERVER_LOGS-my-resource-group",
        "logPath": "/sybase/SY2/ASE-*/install/SY2_BS.log",
        "logType": "SAP_ASE_BACKUP_SERVER_LOGS",
        "monitor": true,
        "encoding": "utf-8"
      },
      {
        "logGroupName": "SAP_ASE_REP_SERVER_LOGS-my-resource-group",
        "logPath": "/sybase/SY2/DM/repservername/repservername.log",
        "logType": "SAP_ASE_REP_SERVER_LOGS",
        "monitor": true,
        "encoding": "utf-8"
      },
      {
        "logGroupName": "SAP_ASE_RMA_AGENT_LOGS-my-resource-group",
        "logPath": "/sybase/SY2/DM/RMA-*/instances/AgentContainer/logs/",
        "logType": "SAP_ASE_RMA_AGENT_LOGS",
        "monitor": true,
        "encoding": "utf-8"
      },
      {
```

```
          "logGroupName": "SAP_ASE_FAULT_MANAGER_LOGS-my-resource-group",
          "logPath": "/opt/sap/FaultManager/dev_sybdbfm",
          "logType": "SAP_ASE_FAULT_MANAGER_LOGS",
          "monitor": true,
          "encoding": "utf-8"
        }
  ],
  "sapAsePrometheusExporter": {
    "sapAseSid": "ASE",
    "sapAsePort": "4901",
    "sapAseSecretName": "ASE_DB_CREDS",
    "prometheusPort": "9399",
    "agreeToEnableASEMonitoring": true
}
```

SAP HANA on Amazon EC2

The following example shows a component configuration in JSON format for SAP HANA on Amazon EC2.

```
{
  "subComponents": [
    {
      "subComponentType": "AWS::EC2::Instance",
      "alarmMetrics": [
        {
          "alarmMetricName": "hanadb_server_startup_time_variations_seconds",
          "monitor": true
        },
        {
          "alarmMetricName": "hanadb_level_5_alerts_count",
          "monitor": true
        },
        {
          "alarmMetricName": "hanadb_level_4_alerts_count",
          "monitor": true
        },
        {
          "alarmMetricName": "hanadb_out_of_memory_events_count",
          "monitor": true
        },
        {
          "alarmMetricName": "hanadb_max_trigger_read_ratio_percent",
```

```
          "monitor": true
        },
        {
          "alarmMetricName": "hanadb_table_allocation_limit_used_percent",
          "monitor": true
        },
        {
          "alarmMetricName": "hanadb_cpu_usage_percent",
          "monitor": true
        },
        {
          "alarmMetricName": "hanadb_plan_cache_hit_ratio_percent",
          "monitor": true
        },
        {
          "alarmMetricName": "hanadb_last_data_backup_age_days",
          "monitor": true
        }
      ],
      "logs": [
        {
          "logGroupName": "SAP_HANA_TRACE-my-resourge-group",
          "logPath": "/usr/sap/HDB/HDB00/*/trace/*.trc",
          "logType": "SAP_HANA_TRACE",
          "monitor": true,
          "encoding": "utf-8"
        },
        {
          "logGroupName": "SAP_HANA_LOGS-my-resource-group",
          "logPath": "/usr/sap/HDB/HDB00/*/trace/*.log",
          "logType": "SAP_HANA_LOGS",
          "monitor": true,
          "encoding": "utf-8"
        }
      ]
    }
  ],
  "hanaPrometheusExporter": {
    "hanaSid": "HDB",
    "hanaPort": "30013",
    "hanaSecretName": "HANA_DB_CREDS",
    "prometheusPort": "9668"
  }
```

```
}
```

SAP HANA High Availability on Amazon EC2

The following example shows a component configuration in JSON format for SAP HANA High Availability on Amazon EC2.

```
{
  "subComponents": [
    {
      "subComponentType": "AWS::EC2::Instance",
      "alarmMetrics": [
        {
          "alarmMetricName": "hanadb_server_startup_time_variations_seconds",
          "monitor": true
        },
        {
          "alarmMetricName": "hanadb_level_5_alerts_count",
          "monitor": true
        },
        {
          "alarmMetricName": "hanadb_level_4_alerts_count",
          "monitor": true
        },
        {
          "alarmMetricName": "hanadb_out_of_memory_events_count",
          "monitor": true
        },
        {
          "alarmMetricName": "ha_cluster_pacemaker_stonith_enabled",
          "monitor": true
        }
      ],
      "logs": [
        {
          "logGroupName": "SAP_HANA_TRACE-my-resourge-group",
          "logPath": "/usr/sap/HDB/HDB00/*/trace/*.trc",
          "logType": "SAP_HANA_TRACE",
          "monitor": true,
          "encoding": "utf-8"
        },
        {
          "logGroupName": "SAP_HANA_HIGH_AVAILABILITY-my-resource-group",
```

```
          "logPath": "/var/log/pacemaker/pacemaker.log",
          "logType": "SAP_HANA_HIGH_AVAILABILITY",
          "monitor": true,
          "encoding": "utf-8"
        }
      ]
    }
  ],
  "hanaPrometheusExporter": {
    "hanaSid": "HDB",
    "hanaPort": "30013",
    "hanaSecretName": "HANA_DB_CREDS",
    "prometheusPort": "9668"
  },
  "haClusterPrometheusExporter": {
    "prometheusPort": "9664"
  }
}
```

SAP NetWeaver on Amazon EC2

The following example shows a component configuration in JSON format for SAP NetWeaver on Amazon EC2.

```
{
  "subComponents": [
    {
      "subComponentType": "AWS::EC2::Instance",
      "alarmMetrics": [
        {
          "alarmMetricName": "CPUUtilization",
          "monitor": true
        },
        {
          "alarmMetricName": "StatusCheckFailed",
          "monitor": true
        },
        {
          "alarmMetricName": "disk_used_percent",
          "monitor": true
        },
        {
          "alarmMetricName": "mem_used_percent",
```

```
            "monitor": true
          },
          {
            "alarmMetricName": "sap_alerts_ResponseTime",
            "monitor": true
          },
          {
            "alarmMetricName": "sap_alerts_ResponseTimeDialog",
            "monitor": true
          },
          {
            "alarmMetricName": "sap_alerts_ResponseTimeDialogRFC",
            "monitor": true
          },
          {
            "alarmMetricName": "sap_alerts_DBRequestTime",
            "monitor": true
          },
          {
            "alarmMetricName": "sap_alerts_LongRunners",
            "monitor": true
          },
          {
            "alarmMetricName": "sap_alerts_AbortedJobs",
            "monitor": true
          },
          {
            "alarmMetricName": "sap_alerts_BasisSystem",
            "monitor": true
          },
          {
            "alarmMetricName": "sap_alerts_Database",
            "monitor": true
          },
          {
            "alarmMetricName": "sap_alerts_Security",
            "monitor": true
          },
          {
            "alarmMetricName": "sap_alerts_System",
            "monitor": true
          },
          {
            "alarmMetricName": "sap_alerts_QueueTime",
```

```
            "monitor": true
          },
          {
            "alarmMetricName": "sap_alerts_Availability",
            "monitor": true
          },
          {
            "alarmMetricName": "sap_start_service_processes",
            "monitor": true
          },
          {
            "alarmMetricName": "sap_dispatcher_queue_now",
            "monitor": true
          },
          {
            "alarmMetricName": "sap_dispatcher_queue_max",
            "monitor": true
          },
          {
            "alarmMetricName": "sap_enqueue_server_locks_max",
            "monitor": true
          },
          {
            "alarmMetricName": "sap_enqueue_server_locks_now",
            "monitor": true
          },
          {
            "alarmMetricName": "sap_enqueue_server_locks_state",
            "monitor": true
          },
          {
            "alarmMetricName": "sap_enqueue_server_replication_state",
            "monitor": true
          }
        ],
        "logs": [
          {
            "logGroupName": "SAP_NETWEAVER_DEV_TRACE_LOGS-NetWeaver-ML4",
            "logPath": "/usr/sap/ML4/*/work/dev_w*",
            "logType": "SAP_NETWEAVER_DEV_TRACE_LOGS",
            "monitor": true,
            "encoding": "utf-8"
          }
        ]
```

```
    }
  ],
  "netWeaverPrometheusExporter": {
    "sapSid": "ML4",
    "instanceNumbers": [
      "00",
      "11"
    ],
    "prometheusPort": "9680"
  }
}
```

SAP NetWeaver High Availability on Amazon EC2

The following example shows a component configuration in JSON format for SAP NetWeaver High Availability on Amazon EC2.

```
{
  "subComponents": [
    {
      "subComponentType": "AWS::EC2::Instance",
      "alarmMetrics": [
        {
          "alarmMetricName": "ha_cluster_corosync_ring_errors",
          "monitor": true
        },
        {
          "alarmMetricName": "ha_cluster_pacemaker_fail_count",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_HA_check_failover_config_state",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_HA_get_failover_config_HAActive",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_alerts_AbortedJobs",
          "monitor": true
        },
        {
```

```
            "alarmMetricName": "sap_alerts_Availability",
            "monitor": true
          },
          {
            "alarmMetricName": "sap_alerts_BasisSystem",
            "monitor": true
          },
          {
            "alarmMetricName": "sap_alerts_DBRequestTime",
            "monitor": true
          },
          {
            "alarmMetricName": "sap_alerts_Database",
            "monitor": true
          },
          {
            "alarmMetricName": "sap_alerts_FrontendResponseTime",
            "monitor": true
          },
          {
            "alarmMetricName": "sap_alerts_LongRunners",
            "monitor": true
          },
          {
            "alarmMetricName": "sap_alerts_QueueTime",
            "monitor": true
          },
          {
            "alarmMetricName": "sap_alerts_ResponseTime",
            "monitor": true
          },
          {
            "alarmMetricName": "sap_alerts_ResponseTimeDialog",
            "monitor": true
          },
          {
            "alarmMetricName": "sap_alerts_ResponseTimeDialogRFC",
            "monitor": true
          },
          {
            "alarmMetricName": "sap_alerts_Security",
            "monitor": true
          },
          {
```

```
          "alarmMetricName": "sap_alerts_Shortdumps",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_alerts_SqlError",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_alerts_System",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_enqueue_server_replication_state",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_start_service_processes",
          "monitor": true
        }
      ],
      "logs": [
        {
          "logGroupName": "SAP_NETWEAVER_DEV_TRACE_LOGS-NetWeaver-PR1",
          "logPath": "/usr/sap/<SID>/D*/work/dev_w*",
          "logType": "SAP_NETWEAVER_DEV_TRACE_LOGS",
          "monitor": true,
          "encoding": "utf-8"
        }
      ]
    }
  ],
  "haClusterPrometheusExporter": {
    "prometheusPort": "9664"
  },
  "netWeaverPrometheusExporter": {
    "sapSid": "PR1",
    "instanceNumbers": [
      "11",
      "12"
    ],
    "prometheusPort": "9680"
  }
}
```

SQL Always On Availability Group

The following example shows a component configuration in JSON format for SQL Always On Availability Group.


```
{
  "subComponents" : [ {
    "subComponentType" : "AWS::EC2::Instance",
    "alarmMetrics" : [ {
      "alarmMetricName" : "CPUUtilization",
      "monitor" : true
    }, {
      "alarmMetricName" : "StatusCheckFailed",
      "monitor" : true
    }, {
      "alarmMetricName" : "Processor % Processor Time",
      "monitor" : true
    }, {
      "alarmMetricName" : "Memory % Committed Bytes In Use",
      "monitor" : true
    }, {
      "alarmMetricName" : "Memory Available Mbytes",
      "monitor" : true
    }, {
      "alarmMetricName" : "Paging File % Usage",
      "monitor" : true
    }, {
      "alarmMetricName" : "System Processor Queue Length",
      "monitor" : true
    }, {
      "alarmMetricName" : "Network Interface Bytes Total/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "PhysicalDisk % Disk Time",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Buffer Manager Buffer cache hit ratio",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Buffer Manager Page life expectancy",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:General Statistics Processes blocked",
      "monitor" : true
```

```
    }, {
      "alarmMetricName" : "SQLServer:General Statistics User Connections",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Locks Number of Deadlocks/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:SQL Statistics Batch Requests/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica File Bytes Received/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Log Bytes Received/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Log remaining for undo",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Log Send Queue",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Mirrored Write Transaction/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Recovery Queue",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Redo Bytes Remaining",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Redone Bytes/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Total Log requiring undo",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Transaction Delay",
      "monitor" : true
    } ],
    "windowsEvents" : [ {
      "logGroupName" : "WINDOWS_EVENTS-Application-<RESOURCE_GROUP_NAME>",
      "eventName" : "Application",
      "eventLevels" : [ "WARNING", "ERROR", "CRITICAL", "INFORMATION" ],
```

```
        "monitor" : true
      }, {
        "logGroupName" : "WINDOWS_EVENTS-System-<RESOURCE_GROUP_NAME>",
        "eventName" : "System",
        "eventLevels" : [ "WARNING", "ERROR", "CRITICAL" ],
        "monitor" : true
      }, {
        "logGroupName" : "WINDOWS_EVENTS-Security-<RESOURCE_GROUP_NAME>",
        "eventName" : "Security",
        "eventLevels" : [ "WARNING", "ERROR", "CRITICAL" ],
        "monitor" : true
      } ],
      "logs" : [ {
        "logGroupName" : "SQL_SERVER_ALWAYSON_AVAILABILITY_GROUP-<RESOURCE_GROUP_NAME>",
        "logPath" : "C:\\Program Files\\Microsoft SQL Server\\MSSQL**.MSSQLSERVER\\MSSQL\
\Log\\ERRORLOG",
        "logType" : "SQL_SERVER",
        "monitor" : true,
        "encoding" : "utf-8"
      } ]
    }, {
      "subComponentType" : "AWS::EC2::Volume",
      "alarmMetrics" : [ {
        "alarmMetricName" : "VolumeReadBytes",
        "monitor" : true
      }, {
      "alarmMetricName" : "VolumeWriteBytes",
        "monitor" : true
      }, {
      "alarmMetricName" : "VolumeReadOps",
        "monitor" : true
      }, {
      "alarmMetricName" : "VolumeWriteOps",
        "monitor" : true
      }, {
      "alarmMetricName" : "VolumeQueueLength",
        "monitor" : true
      }, {
      "alarmMetricName" : "VolumeThroughputPercentage",
        "monitor" : true
      }, {
      "alarmMetricName" : "BurstBalance",
        "monitor" : true
      } ]
```

```
  } ]
}
```

SQL failover cluster instance

The following example shows a component configuration in JSON format for SQL failover cluster instance.

```
{
  "subComponents" : [ {
    "subComponentType" : "AWS::EC2::Instance",
    "alarmMetrics" : [ {
      "alarmMetricName" : "CPUUtilization",
      "monitor" : true
    }, {
      "alarmMetricName" : "StatusCheckFailed",
      "monitor" : true
    }, {
      "alarmMetricName" : "Processor % Processor Time",
      "monitor" : true
    }, {
      "alarmMetricName" : "Memory % Committed Bytes In Use",
      "monitor" : true
    }, {
      "alarmMetricName" : "Memory Available Mbytes",
      "monitor" : true
    }, {
      "alarmMetricName" : "Paging File % Usage",
      "monitor" : true
    }, {
      "alarmMetricName" : "System Processor Queue Length",
      "monitor" : true
    }, {
      "alarmMetricName" : "Network Interface Bytes Total/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "PhysicalDisk % Disk Time",
      "monitor" : true
    }, {
      "alarmMetricName" : "Bytes Received/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "Normal Messages Queue Length/sec",
```

```
        "monitor" : true
      }, {
        "alarmMetricName" : "Urgent Message Queue Length/se",
        "monitor" : true
      }, {
        "alarmMetricName" : "Reconnect Count",
        "monitor" : true
      }, {
        "alarmMetricName" : "Unacknowledged Message Queue Length/sec",
        "monitor" : true
      }, {
        "alarmMetricName" : "Messages Outstanding",
        "monitor" : true
      }, {
        "alarmMetricName" : "Messages Sent/sec",
        "monitor" : true
      }, {
        "alarmMetricName" : "Database Update Messages/sec",
        "monitor" : true
      }, {
        "alarmMetricName" : "Update Messages/sec",
        "monitor" : true
      }, {
        "alarmMetricName" : "Flushes/sec",
        "monitor" : true
      }, {
        "alarmMetricName" : "Crypto Checkpoints Saved/sec",
        "monitor" : true
      }, {
        "alarmMetricName" : "Crypto Checkpoints Restored/sec",
        "monitor" : true
      }, {
        "alarmMetricName" : "Registry Checkpoints Restored/sec",
        "monitor" : true
      }, {
        "alarmMetricName" : "Registry Checkpoints Saved/sec",
        "monitor" : true
      }, {
        "alarmMetricName" : "Cluster API Calls/sec",
        "monitor" : true
      }, {
        "alarmMetricName" : "Resource API Calls/sec",
        "monitor" : true
      }, {
```

```
      "alarmMetricName" : "Cluster Handles/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "Resource Handles/sec",
      "monitor" : true
    } ],
    "windowsEvents" : [ {
      "logGroupName" : "WINDOWS_EVENTS-Application-<RESOURCE_GROUP_NAME>",
      "eventName" : "Application",
      "eventLevels" : [ "WARNING", "ERROR", "CRITICAL"],
      "monitor" : true
    }, {
      "logGroupName" : "WINDOWS_EVENTS-System-<RESOURCE_GROUP_NAME>",
      "eventName" : "System",
      "eventLevels" : [ "WARNING", "ERROR", "CRITICAL", "INFORMATION" ],
      "monitor" : true
    }, {
      "logGroupName" : "WINDOWS_EVENTS-Security-<RESOURCE_GROUP_NAME>",
      "eventName" : "Security",
      "eventLevels" : [ "WARNING", "ERROR", "CRITICAL" ],
      "monitor" : true
    } ],
    "logs" : [ {
      "logGroupName" : "SQL_SERVER_FAILOVER_CLUSTER_INSTANCE-<RESOURCE_GROUP_NAME>",
      "logPath" : "\\\\amznfsxjmzbykwn.mydomain.aws\\SQLDB\\MSSQL**.MSSQLSERVER\\MSSQL\
\Log\\ERRORLOG",
      "logType" : "SQL_SERVER",
      "monitor" : true,
      "encoding" : "utf-8"
    } ]
  }, {
    "subComponentType" : "AWS::EC2::Volume",
    "alarmMetrics" : [ {
      "alarmMetricName" : "VolumeReadBytes",
      "monitor" : true
    }, {
    "alarmMetricName" : "VolumeWriteBytes",
      "monitor" : true
    }, {
    "alarmMetricName" : "VolumeReadOps",
      "monitor" : true
    }, {
    "alarmMetricName" : "VolumeWriteOps",
      "monitor" : true
```

```
    }, {
    "alarmMetricName" : "VolumeQueueLength",
      "monitor" : true
    }, {
    "alarmMetricName" : "VolumeThroughputPercentage",
      "monitor" : true
    }, {
    "alarmMetricName" : "BurstBalance",
      "monitor" : true
    } ]
  } ]
}
```

Create and configure CloudWatch Application Insights monitoring using CloudFormation templates

You can add Application Insights monitoring, including key metrics and telemetry, to your application, database, and web server, directly from Amazon CloudFormation templates.

This section provides sample Amazon CloudFormation templates in both JSON and YAML formats to help you create and configure Application Insights monitoring.

To view the Application Insights resource and property reference in the *Amazon CloudFormation User Guide*, see ApplicationInsights resource type reference.

Sample templates

- Create an Application Insights application for the entire Amazon CloudFormation stack
- Create an Application Insights application with detailed settings
- Create an Application Insights application with CUSTOM mode component configuration
- Create an Application Insights application with DEFAULT mode component configuration
- Create an Application Insights application with DEFAULT_WITH_OVERWRITE mode component configuration

Create an Application Insights application for the entire Amazon CloudFormation stack

To apply the following template, you must create Amazon resources and one or more resource groups from which to create Application Insights applications to monitor those resources. For more information, see Getting started with Amazon Resource Groups.

The first two parts of the following template specify a resource and a resource group. The last part of the template creates an Application Insights application for the resource group, but does not configure the application or apply monitoring. For more information, see the CreateApplication command details in the *Amazon CloudWatch Application Insights API Reference*.

Template in JSON format

```
{
    "AWSTemplateFormatVersion": "2010-09-09",
    "Description": "Test Resource Group stack",
    "Resources": {
        "EC2Instance": {
            "Type": "AWS::EC2::Instance",
            "Properties": {
                "ImageId" : "ami-abcd1234efgh5678i",
                "SecurityGroupIds" : ["sg-abcd1234"]
            }
        },
        ...
        "ResourceGroup": {
            "Type": "AWS::ResourceGroups::Group",
            "Properties": {
                "Name": "my_resource_group"
            }
        },
        "AppInsightsApp": {
            "Type": "AWS::ApplicationInsights::Application",
            "Properties": {
                "ResourceGroupName": "my_resource_group"
            },
            "DependsOn" : "ResourceGroup"
        }
    }
}
```

Template in YAML format

```
---
AWSTemplateFormatVersion: '2010-09-09'
Description: Test Resource Group stack
Resources:
  EC2Instance:
    Type: AWS::EC2::Instance
    Properties:
      ImageId: ami-abcd1234efgh5678i
      SecurityGroupIds:
      - sg-abcd1234
  ...
  ResourceGroup:
    Type: AWS::ResourceGroups::Group
    Properties:
      Name: my_resource_group
  AppInsightsApp:
    Type: AWS::ApplicationInsights::Application
    Properties:
      ResourceGroupName: my_resource_group
    DependsOn: ResourceGroup
```

The following template section applies the default monitoring configuration to the Application Insights application. For more information, see the CreateApplication command details in the *Amazon CloudWatch Application Insights API Reference*.

When `AutoConfigurationEnabled` is set to `true`, all components of the application are configured with the recommended monitoring settings for the `DEFAULT` application tier. For more information about these settings and tiers, see DescribeComponentConfigurationRecommendation and UpdateComponentConfiguration in the *Amazon CloudWatch Application Insights API Reference*.

Template in JSON format

```
{
    "AWSTemplateFormatVersion": "2010-09-09",
    "Description": "Test Application Insights Application stack",
    "Resources": {
        "AppInsightsApp": {
            "Type": "AWS::ApplicationInsights::Application",
            "Properties": {
                "ResourceGroupName": "my_resource_group",
```

```
                "AutoConfigurationEnabled": true
            }
        }
    }
}
```

Template in YAML format

```
---
AWSTemplateFormatVersion: '2010-09-09'
Description: Test Application Insights Application stack
Resources:
  AppInsightsApp:
    Type: AWS::ApplicationInsights::Application
    Properties:
      ResourceGroupName: my_resource_group
      AutoConfigurationEnabled: true
```

Create an Application Insights application with detailed settings

The following template performs these actions:

- Creates an Application Insights application with CloudWatch Events notification and OpsCenter enabled. For more information, see the CreateApplication command details in the *Amazon CloudWatch Application Insights API Reference*.
- Tags the application with two tags, one of which has no tag values. For more information, see TagResource in the *Amazon CloudWatch Application Insights API Reference*.
- Creates two custom instance group components. For more information, see CreateComponent in the *Amazon CloudWatch Application Insights API Reference*.
- Creates two log pattern sets. For more information, see CreateLogPattern in the *Amazon CloudWatch Application Insights API Reference*.
- Sets `AutoConfigurationEnabled` to `true`, which configures all components of the application with the recommended monitoring settings for the `DEFAULT` tier. For more information, see DescribeComponentConfigurationRecommendation in the *Amazon CloudWatch Application Insights API Reference*.

Template in JSON format

```
{
```

```
    "Type": "AWS::ApplicationInsights::Application",
    "Properties": {
        "ResourceGroupName": "my_resource_group",
        "CWEMonitorEnabled": true,
        "OpsCenterEnabled": true,
        "OpsItemSNSTopicArn": "arn:aws-cn:sns:cn-north-1:123456789012:my_topic",
        "AutoConfigurationEnabled": true,
        "Tags": [
            {
                "Key": "key1",
                "Value": "value1"
            },
            {
                "Key": "key2",
                "Value": ""
            }
        ],
        "CustomComponents": [
            {
                "ComponentName": "test_component_1",
                "ResourceList": [
                    "arn:aws-cn:ec2:cn-north-1:123456789012:instance/i-
abcd1234efgh5678i"
                ]
            },
            {
                "ComponentName": "test_component_2",
                "ResourceList": [
                    "arn:aws-cn:ec2:cn-north-1:123456789012:instance/i-
abcd1234efgh5678i",
                    "arn:aws-cn:ec2:cn-north-1:123456789012:instance/i-
abcd1234efgh5678i"
                ]
            }
        ],
        "LogPatternSets": [
            {
                "PatternSetName": "pattern_set_1",
                "LogPatterns": [
                    {
                        "PatternName": "deadlock_pattern",
                        "Pattern": ".*\\sDeadlocked\\sSchedulers(([^\\w].*)|($))",
                        "Rank": 1
                    }
```

```
                ]
            },
            {
                "PatternSetName": "pattern_set_2",
                "LogPatterns": [
                    {
                        "PatternName": "error_pattern",
                        "Pattern": ".*[\\s\\[]ERROR[\\s\\]].*",
                        "Rank": 1
                    },
                    {
                        "PatternName": "warning_pattern",
                        "Pattern": ".*[\\s\\[]WARN(ING)?[\\s\\]].*",
                        "Rank": 10
                    }
                ]
            }
        ]
    }
}
```

Template in YAML format

```
---
Type: AWS::ApplicationInsights::Application
Properties:
  ResourceGroupName: my_resource_group
  CWEMonitorEnabled: true
  OpsCenterEnabled: true
  OpsItemSNSTopicArn: arn:aws-cn:sns:cn-north-1:123456789012:my_topic
  AutoConfigurationEnabled: true
  Tags:
  - Key: key1
    Value: value1
  - Key: key2
    Value: ''
  CustomComponents:
  - ComponentName: test_component_1
    ResourceList:
    - arn:aws-cn:ec2:cn-north-1:123456789012:instance/i-abcd1234efgh5678i
  - ComponentName: test_component_2
    ResourceList:
    - arn:aws-cn:ec2:cn-north-1:123456789012:instance/i-abcd1234efgh5678i
```

```
    - arn:aws-cn:ec2:cn-north-1:123456789012:instance/i-abcd1234efgh5678i
  LogPatternSets:
  - PatternSetName: pattern_set_1
    LogPatterns:
    - PatternName: deadlock_pattern
      Pattern: ".*\\sDeadlocked\\sSchedulers(([^\\w].*)|($))"
      Rank: 1
  - PatternSetName: pattern_set_2
    LogPatterns:
    - PatternName: error_pattern
      Pattern: ".*[\\s\\[]ERROR[\\s\\]].*"
      Rank: 1
    - PatternName: warning_pattern
      Pattern: ".*[\\s\\[]WARN(ING)?[\\s\\]].*"
      Rank: 10
```

Create an Application Insights application with CUSTOM mode component configuration

The following template performs these actions:

- Creates an Application Insights application. For more information, see CreateApplication in the *Amazon CloudWatch Application Insights API Reference*.
- Component `my_component` sets `ComponentConfigurationMode` to `CUSTOM`, which causes this component to be configured with the configuration specified in `CustomComponentConfiguration`. For more information, see UpdateComponentConfiguration in the *Amazon CloudWatch Application Insights API Reference*.

Template in JSON format

```
{
    "Type": "AWS::ApplicationInsights::Application",
    "Properties": {
        "ResourceGroupName": "my_resource_group,
        "ComponentMonitoringSettings": [
            {
                "ComponentARN": "my_component",
                "Tier": "SQL_SERVER",
                "ComponentConfigurationMode": "CUSTOM",
                "CustomComponentConfiguration": {
                    "ConfigurationDetails": {
```

```
                    "AlarmMetrics": [
                        {
                            "AlarmMetricName": "StatusCheckFailed"
                        },
                        ...
                    ],
                    "Logs": [
                        {
                            "LogGroupName": "my_log_group_1",
                            "LogPath": "C:\\LogFolder_1\\*",
                            "LogType": "DOT_NET_CORE",
                            "Encoding": "utf-8",
                            "PatternSet": "my_pattern_set_1"
                        },
                        ...
                    ],
                    "WindowsEvents": [
                        {
                            "LogGroupName": "my_windows_event_log_group_1",
                            "EventName": "Application",
                            "EventLevels": [
                                "ERROR",
                                "WARNING",
                                ...
                            ],
                            "Encoding": "utf-8",
                            "PatternSet": "my_pattern_set_2"
                        },
                        ...
                    ],
                    "Alarms": [
                        {
                            "AlarmName": "my_alarm_name",
                            "Severity": "HIGH"
                        },
                        ...
                    ]
                },
                "SubComponentTypeConfigurations": [
                    {
                        "SubComponentType": "EC2_INSTANCE",
                        "SubComponentConfigurationDetails": {
                            "AlarmMetrics": [
                                {
```

```
                            "AlarmMetricName": "DiskReadOps"
                        },
                        ...
                    ],
                    "Logs": [
                        {
                            "LogGroupName": "my_log_group_2",
                            "LogPath": "C:\\LogFolder_2\\*",
                            "LogType": "IIS",
                            "Encoding": "utf-8",
                            "PatternSet": "my_pattern_set_3"
                        },
                        ...
                    ],
                    "processes" : [
                        {
                            "processName" : "my_process",
                            "alarmMetrics" : [
                        {
                            "alarmMetricName" : "procstat cpu_usage",
                            "monitor" : true
                        }, {
                            "alarmMetricName" : "procstat memory_rss",
                            "monitor" : true
                        }
                    ]
                }
            ],
                    "WindowsEvents": [
                        {
                            "LogGroupName": "my_windows_event_log_group_2",
                            "EventName": "Application",
                            "EventLevels": [
                                "ERROR",
                                "WARNING",
                                ...
                            ],
                            "Encoding": "utf-8",
                            "PatternSet": "my_pattern_set_4"
                        },
                        ...
                    ]
                }
            }
```

```
                    ]
                }
            }
        ]
    }
}
```

Template in YAML format

```
---
Type: AWS::ApplicationInsights::Application
Properties:
  ResourceGroupName: my_resource_group
  ComponentMonitoringSettings:
  - ComponentARN: my_component
    Tier: SQL_SERVER
    ComponentConfigurationMode: CUSTOM
    CustomComponentConfiguration:
      ConfigurationDetails:
        AlarmMetrics:
        - AlarmMetricName: StatusCheckFailed
        ...
        Logs:
        - LogGroupName: my_log_group_1
          LogPath: C:\LogFolder_1\*
          LogType: DOT_NET_CORE
          Encoding: utf-8
          PatternSet: my_pattern_set_1
        ...
        WindowsEvents:
        - LogGroupName: my_windows_event_log_group_1
          EventName: Application
          EventLevels:
          - ERROR
          - WARNING
          ...
          Encoding: utf-8
          PatternSet: my_pattern_set_2
        ...
        Alarms:
        - AlarmName: my_alarm_name
          Severity: HIGH
        ...
```

```
    SubComponentTypeConfigurations:
    - SubComponentType: EC2_INSTANCE
      SubComponentConfigurationDetails:
        AlarmMetrics:
        - AlarmMetricName: DiskReadOps
        ...
        Logs:
        - LogGroupName: my_log_group_2
          LogPath: C:\LogFolder_2\*
          LogType: IIS
          Encoding: utf-8
          PatternSet: my_pattern_set_3
        ...
        Processes:
        - ProcessName: my_process
          AlarmMetrics:
          - AlarmMetricName: procstat cpu_usage
            ...
          ...
        WindowsEvents:
        - LogGroupName: my_windows_event_log_group_2
          EventName: Application
          EventLevels:
          - ERROR
          - WARNING
          ...
          Encoding: utf-8
          PatternSet: my_pattern_set_4
        ...
```

Create an Application Insights application with DEFAULT mode component configuration

The following template performs these actions:

- Creates an Application Insights application. For more information, see CreateApplication in the *Amazon CloudWatch Application Insights API Reference*.
- Component my_component sets ComponentConfigurationMode to DEFAULT and Tier to SQL_SERVER, which causes this component to be configured with the configuration settings that Application Insights recommends for the SQL_Server tier. For more information, see DescribeComponentConfiguration and UpdateComponentConfiguration in the *Amazon CloudWatch Application Insights API Reference*.

Template in JSON format

```
{
    "Type": "AWS::ApplicationInsights::Application",
    "Properties": {
        "ResourceGroupName": "my_resource_group",
        "ComponentMonitoringSettings": [
            {
                "ComponentARN": "my_component",
                "Tier": "SQL_SERVER",
                "ComponentConfigurationMode": "DEFAULT"
            }
        ]
    }
}
```

Template in YAML format

```
---
Type: AWS::ApplicationInsights::Application
Properties:
  ResourceGroupName: my_resource_group
  ComponentMonitoringSettings:
  - ComponentARN: my_component
    Tier: SQL_SERVER
    ComponentConfigurationMode: DEFAULT
```

Create an Application Insights application with `DEFAULT_WITH_OVERWRITE` mode component configuration

The following template performs these actions:

- Creates an Application Insights application. For more information, see [CreateApplication](#) in the *Amazon CloudWatch Application Insights API Reference*.
- Component `my_component` sets `ComponentConfigurationMode` to `DEFAULT_WITH_OVERWRITE` and `tier` to `DOT_NET_CORE`, which causes this component to be configured with the configuration settings that Application Insights recommends for the `DOT_NET_CORE` tier. Overwritten configuration settings are specified in the `DefaultOverwriteComponentConfiguration`:
  - At the component level `AlarmMetrics` settings are overwritten.

- At the sub-component level, for the EC2_Instance type sub-components, Logs settings are overwritten.

For more information, see UpdateComponentConfiguration in the *Amazon CloudWatch Application Insights API Reference*.

Template in JSON format

```
{
    "Type": "AWS::ApplicationInsights::Application",
    "Properties": {
        "ResourceGroupName": "my_resource_group",
        "ComponentMonitoringSettings": [
            {
                "ComponentName": "my_component",
                "Tier": "DOT_NET_CORE",
                "ComponentConfigurationMode": "DEFAULT_WITH_OVERWRITE",
                "DefaultOverwriteComponentConfiguration": {
                    "ConfigurationDetails": {
                        "AlarmMetrics": [
                            {
                                "AlarmMetricName": "StatusCheckFailed"
                            }
                        ]
                    },
                    "SubComponentTypeConfigurations": [
                        {
                            "SubComponentType": "EC2_INSTANCE",
                            "SubComponentConfigurationDetails": {
                                "Logs": [
                                    {
                                        "LogGroupName": "my_log_group",
                                        "LogPath": "C:\\LogFolder\\*",
                                        "LogType": "IIS",
                                        "Encoding": "utf-8",
                                        "PatternSet": "my_pattern_set"
                                    }
                                ]
                            }
                        }
                    ]
                }
```

```
            }
        ]
    }
}
```

Template in YAML format

```
---
Type: AWS::ApplicationInsights::Application
Properties:
  ResourceGroupName: my_resource_group
  ComponentMonitoringSettings:
  - ComponentName: my_component
    Tier: DOT_NET_CORE
    ComponentConfigurationMode: DEFAULT_WITH_OVERWRITE
    DefaultOverwriteComponentConfiguration:
      ConfigurationDetails:
        AlarmMetrics:
        - AlarmMetricName: StatusCheckFailed
      SubComponentTypeConfigurations:
      - SubComponentType: EC2_INSTANCE
        SubComponentConfigurationDetails:
          Logs:
          - LogGroupName: my_log_group
            LogPath: C:\LogFolder\*
            LogType: IIS
            Encoding: utf-8
            PatternSet: my_pattern_set
```

Tutorial: Set up monitoring for SAP ASE

This tutorial demonstrates how to configure CloudWatch Application Insights to set up monitoring for your SAP ASE databases. You can use CloudWatch Application Insights automatic dashboards to visualize problem details, accelerate troubleshooting, and facilitate mean time to resolution (MTTR) for your SAP ASE databases.

Application Insights for SAP ASE topics

- Supported environments
- Supported operating systems
- Features

- Prerequisites
- Set up monitoring on your SAP ASE database
- Manage monitoring of your SAP ASE database
- Configure the alarm threshold
- View and troubleshoot SAP ASE problems detected by Application Insights
- Troubleshooting Application Insights for SAP ASE

Supported environments

CloudWatch Application Insights supports the deployment of Amazon resources for the following systems and patterns. You provide and install SAP ASE database software and supported SAP application software.

- **One or more SAP ASE databases on a single Amazon EC2 instance** – SAP ASE in a single-node, scale-up architecture.
- **Cross-AZ SAP ASE database high availability setup** – SAP ASE with high availability configured across two Availability Zones using SUSE/RHEL clustering.

> **Note**
>
> CloudWatch Application Insights supports only single SAP system ID (SID) ASE HA environments. If multiple ASE HA SIDs are attached, monitoring will be set up for only the first detected SID.

Supported operating systems

CloudWatch Application Insights for SAP ASE supports x86-64 architecture on the following operating systems:

- SuSE Linux 12 SP4
- SuSE Linux 12 SP5
- SuSE Linux 15
- SuSE Linux 15 SP1

- SuSE Linux 15 SP2
- SuSE Linux 15 SP3
- SuSE Linux 15 SP4
- SuSE Linux 15 SP1 For SAP
- SuSE Linux 15 SP2 For SAP
- SuSE Linux 15 SP3 For SAP
- SuSE Linux 15 SP4 For SAP
- SuSE Linux 12 SP4 For SAP
- SuSE Linux 12 SP5 For SAP
- RedHat Linux 7.6
- RedHat Linux 7.7
- RedHat Linux 7.9
- RedHat Linux 8.1
- RedHat Linux 8.4
- RedHat Linux 8.6

Features

CloudWatch Application Insights for SAP ASE provides the following features:

- Automatic SAP ASE workload detection
- Automatic SAP ASE alarm creation based on static threshold
- Automatic SAP ASE alarm creation based on anomaly detection
- Automatic SAP ASE log pattern recognition
- Health dashboard for SAP ASE
- Problem dashboard for SAP ASE

Prerequisites

You must perform the following prerequisites to configure an SAP ASE database with CloudWatch Application Insights:

- **SAP ASE configuration parameters** – The following configuration parameters must be enabled on your ASE DB: `"enable monitoring"`, `"sql text pipe max messages"`, `"sql text pipe active"`. This allows CloudWatch Application Insights to provide full monitoring capabilities for your DB. If these settings aren't enabled on your ASE database, Application Insights will automatically enable them to collect the necessary metrics to allow monitoring.
- **SAP ASE database user** – The database user provided during Application Insights onboarding must have permission to access the following:
  - System tables in the master database and user (tenant) databases
  - Monitoring tables
- **SAPHostCtrl** – Install and set up SAPHostCtrl on your Amazon EC2 instance.
- **Amazon CloudWatch agent** – Make sure that you are not running a preexisting CloudWatch agent on your Amazon EC2 instance. If you have CloudWatch agent installed, make sure to remove the configuration of the resources you are using in CloudWatch Application Insights from the existing CloudWatch agent configuration file to avoid a merge conflict. For more information, see Manually create or edit the CloudWatch agent configuration file.
- **Amazon Systems Manager enablement** – Install SSM Agent on your instances, and enable the instances enabled for SSM. For information about how to install the SSM agent, see Working with SSM Agent in the *Amazon Systems Manager User Guide*.
- **Amazon EC2 instance roles** – You must attach the following Amazon EC2 instance roles to configure your database.
  - You must attach the `AmazonSSMManagedInstanceCore` role to enable Systems Manager. For more information, see Amazon Systems Manager identity-based policy examples.
  - You must attach the `CloudWatchAgentServerPolicy` to enable instance metrics and logs to be emitted through CloudWatch. For more information, see Create IAM roles and users for use with Amazon CloudWatch agent.
  - You must attach the following IAM inline policy to the Amazon EC2 instance role to read the password stored in Amazon Secrets Manager. For more information about inline policies, see Inline policies in the *Amazon Identity and Access Management User Guide*.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
```

```
                "secretsmanager:GetSecretValue"
            ],
            "Resource": "arn:aws:secretsmanager:*:*:secret:ApplicationInsights-*"
        }
    ]
}
```

- **Amazon Resource Groups** – You must create a resource group that includes all of the associated Amazon resources used by your application stack to onboard your applications to CloudWatch Application Insights. This includes Amazon EC2 instances and Amazon EBS volumes running your SAP ASE database. If there are multiple databases per account, we recommend that you create one resource group that includes the Amazon resources for each SAP ASE database system.
- **IAM permissions** – For non-admin users:
  - You must create an Amazon Identity and Access Management (IAM) policy that allows Application Insights to create a service-linked role, and attach it to your user identity. For steps to attach the policy, see IAM policy for CloudWatch Application Insights.
  - The user must have permission to create a secret in Amazon Secrets Manager to store the database user credentials. For more information, see Example: Permission to create secrets.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "secretsmanager:CreateSecret"
      ],
      "Resource": "arn:aws:secretsmanager:*:*:secret:ApplicationInsights-*"
    }
  ]
}
```

- **Service-linked role** – Application Insights uses Amazon Identity and Access Management (IAM) service-linked roles. A service-linked role is created for you when you create your first Application Insights application in the Application Insights console. For more information, see Using service-linked roles for CloudWatch Application Insights.

Set up monitoring on your SAP ASE database

Use the following steps to set up monitoring for your SAP ASE database

1. Open the CloudWatch console.
2. From the left navigation pane, under **Insights**, choose **Application Insights**.
3. The **Application Insights** page displays the list of applications that are monitored with Application Insights, and the monitoring status for each application. In the upper right-hand corner, choose **Add an application**.
4. On the **Specify application details** page, from the dropdown list under **Resource group**, select the Amazon resource group that contains your SAP ASE database resources. If you haven't created a resource group for your application, you can create one by choosing **Create new resource group** under the **Resource group** dropdown. For more information about creating resource groups, see the *Amazon Resource Groups User Guide.*
5. Under **Monitor CloudWatch Events**, select the check box to integrate Application Insights monitoring with CloudWatch Events to get insights from Amazon EBS, Amazon EC2, Amazon CodeDeploy, Amazon ECS, Amazon Health APIs and notifications, Amazon RDS, Amazon S3, and Amazon Step Functions.
6. Under **Integrate with Amazon Systems Manager OpsCenter**, select the check box next to **Generate Amazon Systems Manager OpsCenter OpsItems for remedial actions** to view and get notifications when problems are detected for the selected applications. To track the operations that are performed to resolve operational work items, called OpsItems, that are related to your Amazon resources, provide an SNS topic ARN.
7. You can optionally enter tags to help you identify and organize your resources. CloudWatch Application Insights supports both tag-based and Amazon CloudFormation stack-based resource groups, with the exception of Application Auto Scaling groups. For more information, see Tag Editor in the *Amazon Resource Groups and Tags User Guide*.
8. Choose **Next** to continue to set up monitoring.
9. On the **Review detected components** page, the monitored components and their workloads automatically detected by CloudWatch Application Insights are listed.

   > **Note**
   >
   > Components that contain a detected SAP ASE High Availability workload support only one workload on a component. Components that contain a detected SAP ASE single node workload support multiple workloads, but you can't add or remove workloads. All automatically detected workloads will be monitored.

10. Choose **Next**.

11. On the **Specify component details** page, enter the username and password of your SAP ASE databases.
12. Review your application monitoring configuration, and choose **Submit**.
13. The application details page opens, where you can view the **Application summary**, the list of **Monitored components and workloads**, and **Unmonitored components and workloads**. If you select the radio button next to a component or workload, you can also view the **Configuration history**, **Log patterns**, and any **Tags** that you have created. When you submit your configuration, your account deploys all of the metrics and alarms for your SAP ASE system, which can take up to 2 hours.

Manage monitoring of your SAP ASE database

You can manage user credentials, metrics, and log paths for your SAP ASE database by performing the following steps:

1. Open the CloudWatch console.
2. From the left navigation pane, under **Insights**, choose **Application Insights**.
3. The **Application Insights** page displays the list of applications that are monitored with Application Insights, and the monitoring status for each application.
4. Under **Monitored components**, select the radio button next to the component name. Then, choose **Manage monitoring**.
5. Under **EC2 instance group logs**, you can update the existing log path, log pattern set, and log group name. In addition, you can add up to three additional **Application logs**.
6. Under **Metrics**, you can choose the SAP ASE metrics according to your requirements. SAP ASE metric names are prefixed with `asedb`. You can add up to 60 metrics per component.
7. Under **ASE configuration**, enter the username and password for the SAP ASE database. This is the username and password that Amazon CloudWatch agent uses to connect to the SAP ASE database.
8. Under **Custom alarms**, you can add additional alarms to be monitored by CloudWatch Application Insights.
9. Review your application monitoring configuration and choose **Submit**. When you submit your configuration, your account updates all of the metrics and alarms for your SAP HANA system, which can take up to 2 hours.

Configure the alarm threshold

CloudWatch Application Insights automatically creates a Amazon CloudWatch metric for the alarm to watch, along with the threshold for that metric. The alarm changes to the `ALARM` state when the metric surpasses the threshold for a specified number of evaluation periods. Note that these settings are not retained by Application Insights.

To edit an alarm for a single metric, perform the following steps:

1. Open the [CloudWatch console](https://console.aws.amazon.com/cloudwatch/).
2. In the left navigation pane, choose **Alarms**>**All alarms**.
3. Select the radio button next to the alarm that was automatically created by CloudWatch Application Insights. Then choose **Actions**, and select **Edit** from the dropdown menu.
4. Edit the following parameters under **Metric**.

    a. Under **Statistic**, choose one of the statistics or predefined percentiles, or specify a custom percentile. For example, `p95.45`.

    b. Under **Period**, choose the evaluations period for the alarm. When you evaluate the alarm, each period is aggregated into one data point.
5. Edit the following parameters under **Conditions**.

    a. Choose whether the metric must be greater than, less than, or equal to the threshold.

    b. Specify the threshold value.
6. Under **Additional configuration** edit the following parameters.

    a. Under **Datapoints to alarm**, specify the number of data points, or evaluation periods, that must be in the `ALARM` state to initiate the alarm. When the two values match, an alarm is created that enters `ALARM` state if the designated number of consecutive periods are exceeded. To create an `m` out of `n` alarm, specify a lower value for the first data point than for the second. For more information about evaluating alarms, see Evaluating an alarm.

    b. Under **Missing data treatment**, choose the behavior of the alarm when some data points are missing. For more information about missing data treatment, see Configuring how CloudWatch alarms treat missing data.

    c. If the alarm uses a percentile as the monitored statistic, a **Percentiles with low samples** box appears. Choose whether to evaluate or ignore cases with low sample rates. If you choose **ignore (maintain alarm state)**, the current alarm state is always maintained when the sample size is too low. For more information about percentiles with low samples, see Percentile-based CloudWatch alarms and low data samples.

7. Choose **Next**.
8. Under **Notification**, select an SNS topic to notify when the alarm is in `ALARM` state, `OK` state, or `INSUFFICIENT_DATA` state.
9. Choose **Update alarm**.

View and troubleshoot SAP ASE problems detected by Application Insights

This section helps you resolve common troubleshooting problems that occur when you configure monitoring for SAP ASE on Application Insights.

SAP ASE Backup Server errors

You can identify the error message by checking the dynamically created dashboard. The dashboard shows the error message reported in the SAP ASE Backup Server. For more details about SAP ASE Backup Server logs, see SAP Documentation Backup Server Error Logging.

SAP ASE long running transactions

Identify the long running transaction and confirm whether it can be stopped or if the running time is intentional. For more details, see 2180410 — How to display transaction log records for long running transactions? — SAP ASE.

SAP ASE User connections

Review whether your SAP ASE database is sized accordingly for the workload you intend to run on the database. For more details, see Configuring User Connections in the SAP documentation.

SAP ASE disk space

You can identify the database layer that is causing the problem by checking the dynamically created dashboard. The dashboard shows the related metrics and log file snippets. It is important to understand the cause of the disk growth and when applicable, increase the physical disk size, the allocated disk space, or both. For more details, see SAP Documentation disk resize in the SAP documentation.

Troubleshooting Application Insights for SAP ASE

This section provides steps to help you resolve common errors returned by the Application Insights dashboard.

Error	Error returned	Root cause	Resolution
Unable to add more than 60 monitor metrics.	`Component cannot have more than 60 monitored metric`	The current metric limit is 60 monitored metrics per component.	Remove unnecessary metrics to adhere to the limit.
No SAP metrics or alarms appear after the onboarding process	The `run` command on the `AWS-ConfigureAWSPackage` failed in Amazon Systems Manager. The output shows the error: `CT-LIBRARY error:ct_connect(): protocol specific layer: external error: The attempt to connect to the server failed`	The username and password might be incorrect.	Verify that the username and password are valid, then rerun the onboarding process.

Tutorial: Set up monitoring for SAP HANA

This tutorial demonstrates how to configure CloudWatch Application Insights to set up monitoring for your SAP HANA databases. You can use CloudWatch Application Insights automatic dashboards to visualize problem details, accelerate troubleshooting, and facilitate mean time to resolution (MTTR) for your SAP HANA databases.

Application Insights for SAP HANA topics

- Supported environments

Supported environments

CloudWatch Application Insights supports the deployment of Amazon resources for the following systems and patterns. You provide and install SAP HANA database software and supported SAP application software.

- **SAP HANA database on a single Amazon EC2 instance** — SAP HANA in a single-node, scale-up architecture, with up to 24TB of memory.
- **SAP HANA database on multiple Amazon EC2 instances** — SAP HANA in a multi-node, scale-out architecture.
- **Cross-AZ SAP HANA database high availability setup** — SAP HANA with high availability configured across two Availability Zones using SUSE/RHEL clustering.

> **Note**
>
> CloudWatch Application Insights supports only single SID HANA environments. If multiple HANA SIDs are attached, monitoring will be set up for only the first detected SID.

Supported operating systems

CloudWatch Application Insights for SAP HANA supports x86-64 architecture on the following operating systems:

- SuSE Linux 12 SP4 For SAP
- SuSE Linux 12 SP5 For SAP

- SuSE Linux 15
- SuSE Linux 15 SP1
- SuSE Linux 15 SP2
- SuSE Linux 15 For SAP
- SuSE Linux 15 SP1 For SAP
- SuSE Linux 15 SP2 For SAP
- SuSE Linux 15 SP3 For SAP
- SuSE Linux 15 SP4 For SAP
- SuSE Linux 15 SP5 For SAP
- RedHat Linux 8.6 For SAP With High Availability and Update Services
- RedHat Linux 8.5 For SAP With High Availability and Update Services
- RedHat Linux 8.4 For SAP With High Availability and Update Services
- RedHat Linux 8.3 For SAP With High Availability and Update Services
- RedHat Linux 8.2 For SAP With High Availability and Update Services
- RedHat Linux 8.1 For SAP With High Availability and Update Services
- RedHat Linux 7.9 For SAP With High Availability and Update Services

Features

CloudWatch Application Insights for SAP HANA provides the following features:

- Automatic SAP HANA workload detection
- Automatic SAP HANA alarm creation based on static threshold
- Automatic SAP HANA alarm creation based on anomaly detection
- Automatic SAP HANA log pattern recognition
- Health dashboard for SAP HANA
- Problem dashboard for SAP HANA

Prerequisites

You must perform the following prerequisites to configure an SAP HANA database with CloudWatch Application Insights:

- **SAP HANA** – Install a running and reachable SAP HANA database 2.0 SPS05 on an Amazon EC2 instance.
- **SAP HANA database user** – A database user with monitoring roles must be created in the SYSTEM database and all tenants.

  Example

  The following SQL commands create a user with monitoring roles.

```
su - <sid>adm
hdbsql -u SYSTEM -p <SYSTEMDB password> -d SYSTEMDB
CREATE USER CW_HANADB_EXPORTER_USER PASSWORD <Monitoring user password> NO
 FORCE_FIRST_PASSWORD_CHANGE;
CREATE ROLE CW_HANADB_EXPORTER_ROLE;
GRANT MONITORING TO CW_HANADB_EXPORTER_ROLE;
GRANT CW_HANADB_EXPORTER_ROLE TO CW_HANADB_EXPORTER_USER;
```

- **Python 3.8** – Install Python 3.8 or later versions on your operating system. Use the latest release of Python. If Python3 is not detected on your operating system, Python 3.6 will be installed.

  For more information, see the installation example.

> **Note**
>
> Manual installation of Python 3.8 or higher is required for SuSE Linux 15 SP4, RedHat Linux 8.6, and later operating systems.

- **Pip3** – Install the installer program, pip3, on your operating system. If pip3 is not detected on your operating system, it will be installed.
- **hdbclient** – CloudWatch Application Insights uses the python driver to connect to the SAP HANA database. If the client is not installed under python3, ensure that you have hdbclient tar file version `2.10 or later` under `/hana/shared/SID/hdbclient/`.
- **Amazon CloudWatch agent** – Make sure that you are not running a preexisting CloudWatch agent on your Amazon EC2 instance. If you have CloudWatch agent installed, make sure to remove the configuration of the resources you are using in CloudWatch Application Insights from the existing CloudWatch agent configuration file to avoid a merge conflict. For more information, see Manually create or edit the CloudWatch agent configuration file.

- **Amazon Systems Manager enablement –** Install SSM Agent on your instances, and the instances must be enabled for SSM. For information about how to install the SSM Agent, see Working with SSM Agent in the *Amazon Systems Manager User Guide*.
- **Amazon EC2 instance roles –** You must attach the following Amazon EC2 instance roles to configure your database.
  - You must attach the `AmazonSSMManagedInstanceCore` role to enable Systems Manager. For more information, see Amazon Systems Manager identity-based policy examples.
  - You must attach the `CloudWatchAgentServerPolicy` to enable instance metrics and logs to be emitted through CloudWatch. For more information, see Create IAM roles and users for use with CloudWatch agent.
  - You must attach the following IAM inline policy to the Amazon EC2 instance role to read the password stored in Amazon Secrets Manager. For more information about inline policies, see Inline policies in the *Amazon Identity and Access Management User Guide*.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue"
            ],
            "Resource": "arn:aws:secretsmanager:*:*:secret:ApplicationInsights-*"
        }
    ]
}
```

- **Amazon resource groups –** You must create a resource group that includes all of the associated Amazon resources used by your application stack to onboard your applications to CloudWatch Application Insights. This includes Amazon EC2 instances and Amazon EBS volumes running your SAP HANA database. If there are multiple databases per account, we recommend that you create one resource group that includes the Amazon resources for each SAP HANA database system.
- **IAM permissions –** For non-admin users:
  - You must create an Amazon Identity and Access Management (IAM) policy that allows Application Insights to create a service-linked role, and attach it to your user identity. For steps to attach the policy, see IAM policy for CloudWatch Application Insights.

- The user must have permission to create a secret in Amazon Secrets Manager to store the database user credentials. For more information, see Example: Permission to create secrets.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "secretsmanager:CreateSecret"
      ],
      "Resource": "arn:aws:secretsmanager:*:*:secret:ApplicationInsights-*"
    }
  ]
}
```

- **Service-linked role** – Application Insights uses Amazon Identity and Access Management (IAM) service-linked roles. A service-linked role is created for you when you create your first Application Insights application in the Application Insights console. For more information, see Using service-linked roles for CloudWatch Application Insights.

Set up your SAP HANA database for monitoring

Use the following steps to set up monitoring for your SAP HANA database

1. Open the CloudWatch console.
2. From the left navigation pane, under **Insights**, choose **Application Insights**.
3. The **Application Insights** page displays the list of applications that are monitored with Application Insights, and the monitoring status for each application. In the upper right-hand corner, choose **Add an application**.
4. On the **Specify application details** page, from the dropdown list under **Resource group**, select the Amazon resource group that contains your SAP HANA database resources. If you haven't created a resource group for your application, you can create one by choosing **Create new resource group** under the **Resource group** dropdown. For more information about creating resource groups, see the *Amazon Resource Groups User Guide.*
5. Under **Monitor CloudWatch Events**, select the check box to integrate Application Insights monitoring with CloudWatch Events to get insights from Amazon EBS, Amazon EC2, Amazon

CodeDeploy, Amazon ECS, Amazon Health APIs and notifications, Amazon RDS, Amazon S3, and Amazon Step Functions.

6. Under **Integrate with Amazon Systems Manager OpsCenter**, select the check box next to **Generate Amazon Systems Manager OpsCenter OpsItems for remedial actions** to view and get notifications when problems are detected for the selected applications. To track the operations that are performed to resolve operational work items, called OpsItems, that are related to your Amazon resources, provide an SNS topic ARN.

7. You can optionally enter tags to help you identify and organize your resources. CloudWatch Application Insights supports both tag-based and Amazon CloudFormation stack-based resource groups, with the exception of Application Auto Scaling groups. For more information, see Tag Editor in the *Amazon Resource Groups and Tags User Guide*.

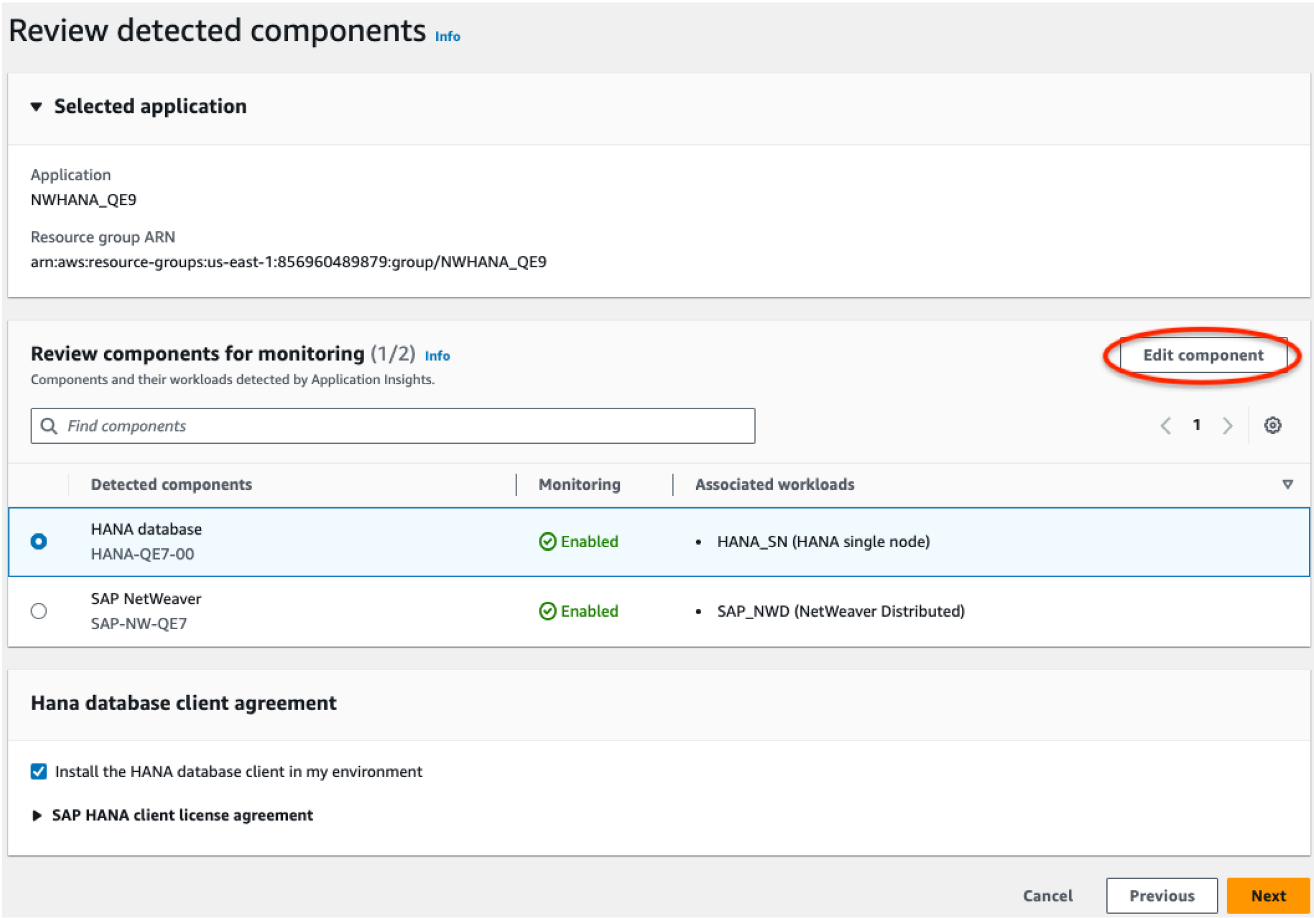
8. Choose **Next** to continue to set up monitoring.

9. On the **Review detected components** page, the monitored components and their workloads automatically detected by CloudWatch Application Insights are listed.

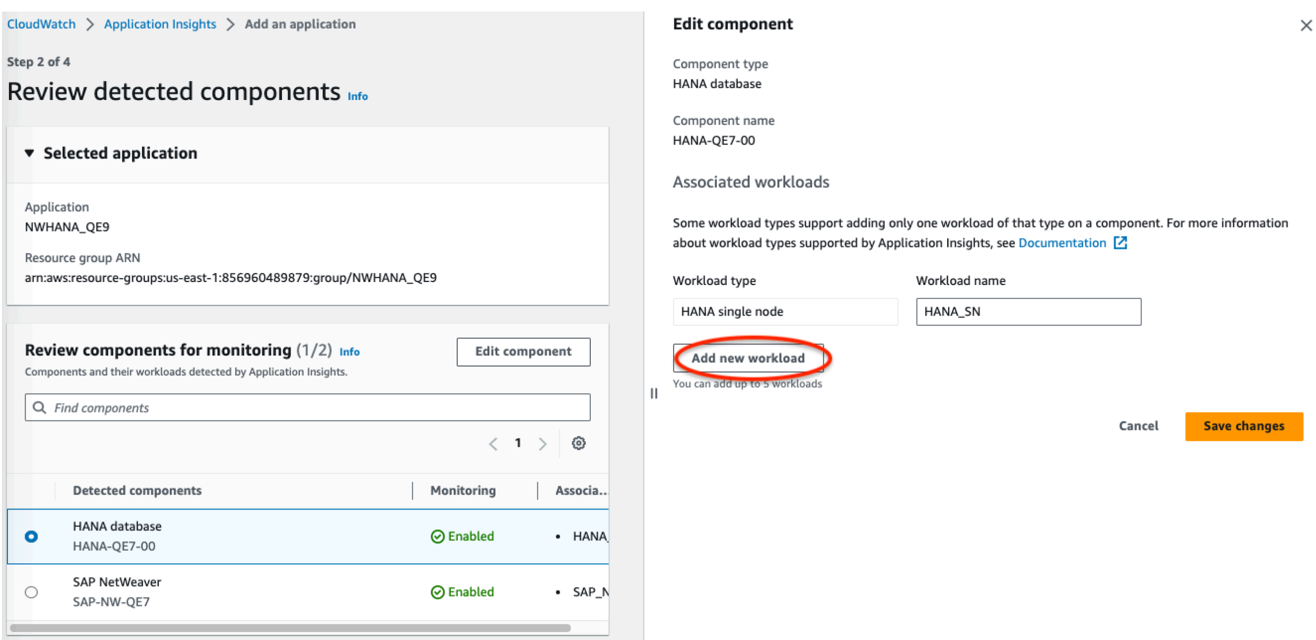
   a. To add workloads to a component that contains a detected SAP HANA single node workload, select the component, then choose **Edit component**.

      Note

      Components that contain a detected SAP HANA multi node or HANA High Availability workload support only one workload on a component.



b. To add a new workload, choose **Add new workload**.



   c. When you are finished editing workloads, choose **Save changes**.

10. Choose **Next**.
11. On the **Specify component details** page, enter the username and password.
12. Review your application monitoring configuration, and choose **Submit**.
13. The application details page opens, where you can view the **Application summary**, the list of **Monitored components and workloads**, and **Unmonitored components and workloads**. If you select the radio button next to a component or workload, you can also view the **Configuration history**, **Log patterns**, and any **Tags** that you have created. When you submit your configuration, your account deploys all of the metrics and alarms for your SAP HANA system, which can take up to 2 hours.

Manage monitoring of your SAP HANA database

You can manage user credentials, metrics, and log paths for your SAP HANA database by performing the following steps:

1. Open the [CloudWatch console](https://console.aws.amazon.com/cloudwatch/).
2. From the left navigation pane, under **Insights**, choose **Application Insights**.
3. The **Application Insights** page displays the list of applications that are monitored with Application Insights, and the monitoring status for each application.
4. Under **Monitored components**, select the radio button next to the component name. Then, choose **Manage monitoring**.
5. Under **EC2 instance group logs**, you can update the existing log path, log pattern set, and log group name. In addition, you can add up to three additional **Application logs**.
6. Under **Metrics**, you can choose the SAP HANA metrics according to your requirements. SAP HANA metric names are prefixed with `hanadb`. You can add up to 40 metrics per component.
7. Under **HANA configuration**, enter the password and user name for the SAP HANA database. This is the username and password that Amazon CloudWatch agent uses to connect to the SAP HANA database.
8. Under **Custom alarms**, you can add additional alarms to be monitored by CloudWatch Application Insights.
9. Review your application monitoring configuration and choose **Submit**. When you submit your configuration, your account updates all of the metrics and alarms for your SAP HANA system, which can take up to 2 hours.

View and troubleshoot SAP HANA problems detected by CloudWatch Application Insights

The following sections provide steps to help you resolve common troubleshooting scenarios that occur when you configure monitoring for SAP HANA on Application Insights.

Troubleshooting topics

SAP HANA database reaches memory allocation limit

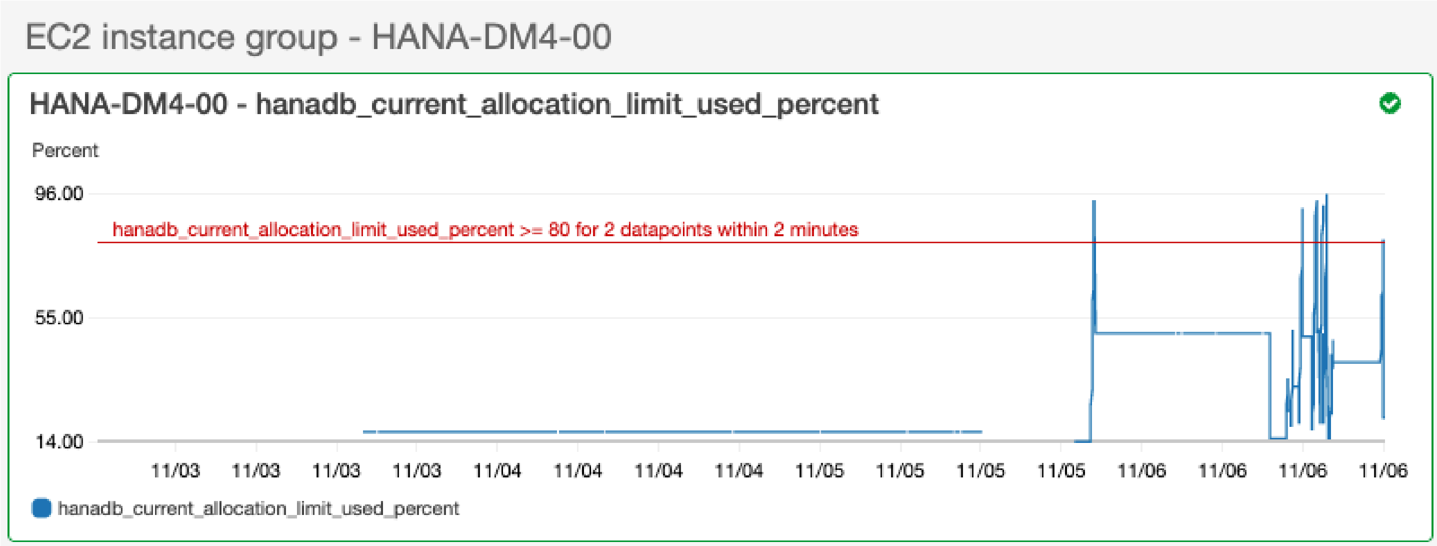
Description

Your SAP application that is backed by an SAP HANA database malfunctions because of high memory pressure, leading to application performance degradation.

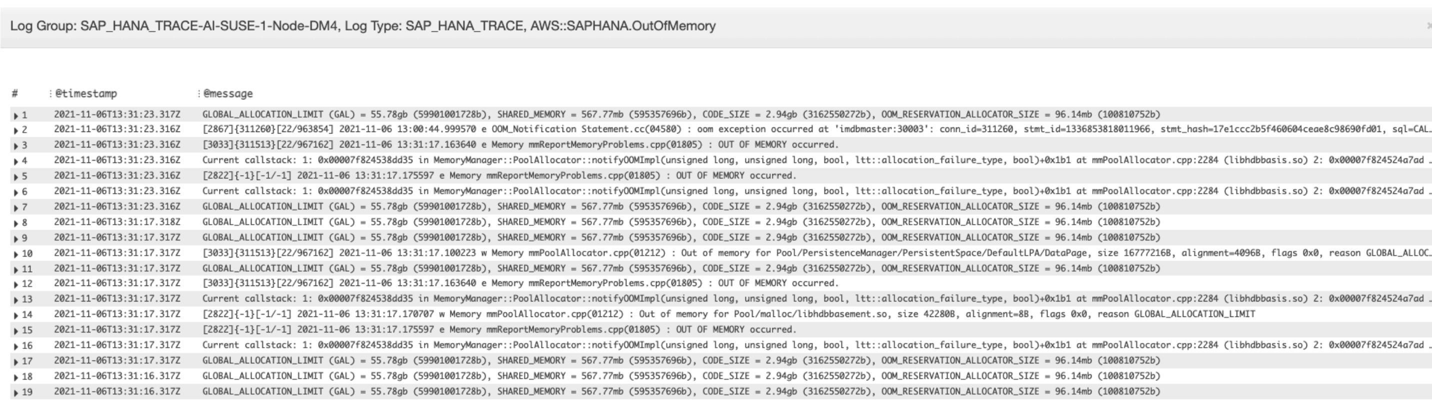
Resolution

You can identify the application layer that is causing the problem by checking the dynamically created dashboard, which shows the related metrics and log file snippets. In the following example, the problem may be because of a large data load in the SAP HANA system.

The used memory allocation exceeds the threshold of 80 percent of the total memory allocation limit.



The log group shows the scheme BNR-DATA and table IMDBMASTER_30003 ran out of memory. In addition, the log group shows the exact time of the issue, current global location limit, shared memory, code size, and OOM reservation allocation size.



Log Group: SAP_HANA_TRACE-AI-SUSE-1-Node-DM4, Log Type: SAP_HANA_TRACE, AWS::SAPHANA.OutOfMemory

#	@timestamp	@message
1	2021-11-06T13:31:23.317Z	GLOBAL_ALLOCATION_LIMIT (GAL) = 55.78gb (59901001728b), SHARED_MEMORY = 567.77mb (595357696b), CODE_SIZE = 2.94gb (3162550272b), OOM_RESERVATION_ALLOCATOR_SIZE = 96.14mb (100810752b)
2	2021-11-06T13:31:23.316Z	[2867]{311260}[22/963854] 2021-11-06 13:00:44.999570 e OOM_Notification Statement.cc(04580) : oom exception occurred at 'imdbmaster:30003': conn_id=311260, stmt_id=1336853818011966, stmt_hash=17e1ccc2b5f460604ceae8c98690fd01, sql=CAL…
3	2021-11-06T13:31:23.316Z	[3033]{311513}[22/967162] 2021-11-06 13:31:17.163640 e Memory mmReportMemoryProblems.cpp(01805) : OUT OF MEMORY occurred.
4	2021-11-06T13:31:23.316Z	Current callstack: 1: 0x00007f824538dd35 in MemoryManager::PoolAllocator::notifyOOMImpl(unsigned long, unsigned long, bool, ltt::allocation_failure_type, bool)+0x1b1 at mmPoolAllocator.cpp:2284 (libhdbbasis.so) 2: 0x00007f824524a7ad …
5	2021-11-06T13:31:23.316Z	[2822]{-1}[-1/-1] 2021-11-06 13:31:17.175597 e Memory mmReportMemoryProblems.cpp(01805) : OUT OF MEMORY occurred.
6	2021-11-06T13:31:23.316Z	Current callstack: 1: 0x00007f824538dd35 in MemoryManager::PoolAllocator::notifyOOMImpl(unsigned long, unsigned long, bool, ltt::allocation_failure_type, bool)+0x1b1 at mmPoolAllocator.cpp:2284 (libhdbbasis.so) 2: 0x00007f824524a7ad …
7	2021-11-06T13:31:23.316Z	GLOBAL_ALLOCATION_LIMIT (GAL) = 55.78gb (59901001728b), SHARED_MEMORY = 567.77mb (595357696b), CODE_SIZE = 2.94gb (3162550272b), OOM_RESERVATION_ALLOCATOR_SIZE = 96.14mb (100810752b)
8	2021-11-06T13:31:17.318Z	GLOBAL_ALLOCATION_LIMIT (GAL) = 55.78gb (59901001728b), SHARED_MEMORY = 567.77mb (595357696b), CODE_SIZE = 2.94gb (3162550272b), OOM_RESERVATION_ALLOCATOR_SIZE = 96.14mb (100810752b)
9	2021-11-06T13:31:17.317Z	GLOBAL_ALLOCATION_LIMIT (GAL) = 55.78gb (59901001728b), SHARED_MEMORY = 567.77mb (595357696b), CODE_SIZE = 2.94gb (3162550272b), OOM_RESERVATION_ALLOCATOR_SIZE = 96.14mb (100810752b)
10	2021-11-06T13:31:17.317Z	[3033]{311513}[22/967162] 2021-11-06 13:31:17.100223 w Memory mmPoolAllocator.cpp(01212) : Out of memory for Pool/PersistenceManager/PersistentSpace/DefaultLPA/DataPage, size 16777216B, alignment=4096B, flags 0x0, reason GLOBAL_ALLOC…
11	2021-11-06T13:31:17.317Z	GLOBAL_ALLOCATION_LIMIT (GAL) = 55.78gb (59901001728b), SHARED_MEMORY = 567.77mb (595357696b), CODE_SIZE = 2.94gb (3162550272b), OOM_RESERVATION_ALLOCATOR_SIZE = 96.14mb (100810752b)
12	2021-11-06T13:31:17.317Z	[3033]{311513}[22/967162] 2021-11-06 13:31:17.163640 e Memory mmReportMemoryProblems.cpp(01805) : OUT OF MEMORY occurred.
13	2021-11-06T13:31:17.317Z	Current callstack: 1: 0x00007f824538dd35 in MemoryManager::PoolAllocator::notifyOOMImpl(unsigned long, unsigned long, bool, ltt::allocation_failure_type, bool)+0x1b1 at mmPoolAllocator.cpp:2284 (libhdbbasis.so) 2: 0x00007f824524a7ad …
14	2021-11-06T13:31:17.317Z	[2822]{-1}[-1/-1] 2021-11-06 13:31:17.170707 w Memory mmPoolAllocator.cpp(01212) : Out of memory for Pool/malloc/libhdbbasement.so, size 42280B, alignment=8B, flags 0x0, reason GLOBAL_ALLOCATION_LIMIT
15	2021-11-06T13:31:17.317Z	[2822]{-1}[-1/-1] 2021-11-06 13:31:17.175597 e Memory mmReportMemoryProblems.cpp(01805) : OUT OF MEMORY occurred.
16	2021-11-06T13:31:17.317Z	Current callstack: 1: 0x00007f824538dd35 in MemoryManager::PoolAllocator::notifyOOMImpl(unsigned long, unsigned long, bool, ltt::allocation_failure_type, bool)+0x1b1 at mmPoolAllocator.cpp:2284 (libhdbbasis.so) 2: 0x00007f824524a7ad …
17	2021-11-06T13:31:17.317Z	GLOBAL_ALLOCATION_LIMIT (GAL) = 55.78gb (59901001728b), SHARED_MEMORY = 567.77mb (595357696b), CODE_SIZE = 2.94gb (3162550272b), OOM_RESERVATION_ALLOCATOR_SIZE = 96.14mb (100810752b)
18	2021-11-06T13:31:16.317Z	GLOBAL_ALLOCATION_LIMIT (GAL) = 55.78gb (59901001728b), SHARED_MEMORY = 567.77mb (595357696b), CODE_SIZE = 2.94gb (3162550272b), OOM_RESERVATION_ALLOCATOR_SIZE = 96.14mb (100810752b)
19	2021-11-06T13:31:16.317Z	GLOBAL_ALLOCATION_LIMIT (GAL) = 55.78gb (59901001728b), SHARED_MEMORY = 567.77mb (595357696b), CODE_SIZE = 2.94gb (3162550272b), OOM_RESERVATION_ALLOCATOR_SIZE = 96.14mb (100810752b)

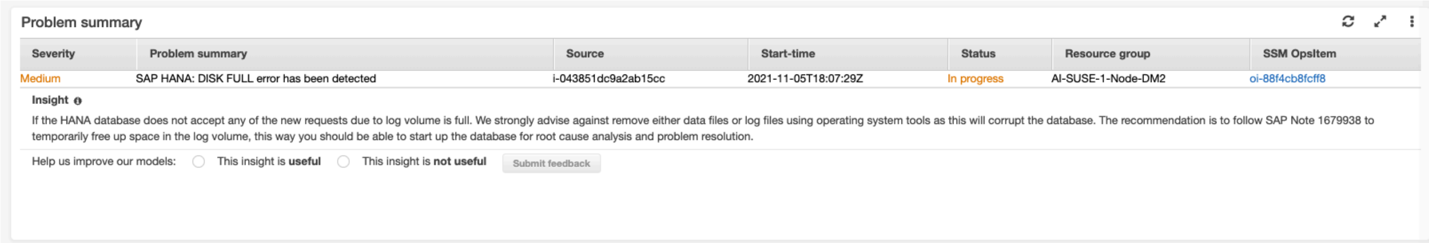
Disk full event

Description

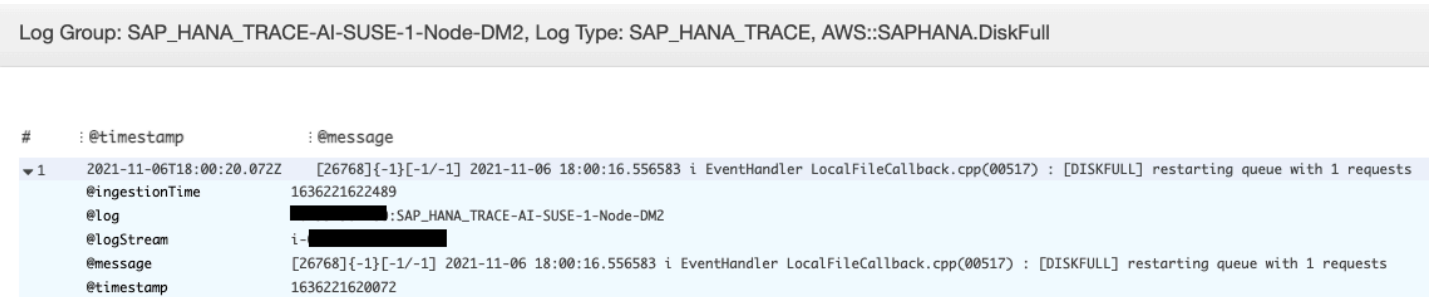
Your SAP application that is backed by an SAP HANA database stops responding, which leads to an inability to access the database.

Resolution

You can identify the database layer that is causing the problem by checking the dynamically created dashboard, which shows the related metrics and log file snippets. In the following example, the problem may be that the administrator failed to enable automatic log backup, which caused the sap/hana/log directory to fill up.



The log group widget in the problem dashboard shows the `DISKFULL` event.



SAP HANA backup stopped running

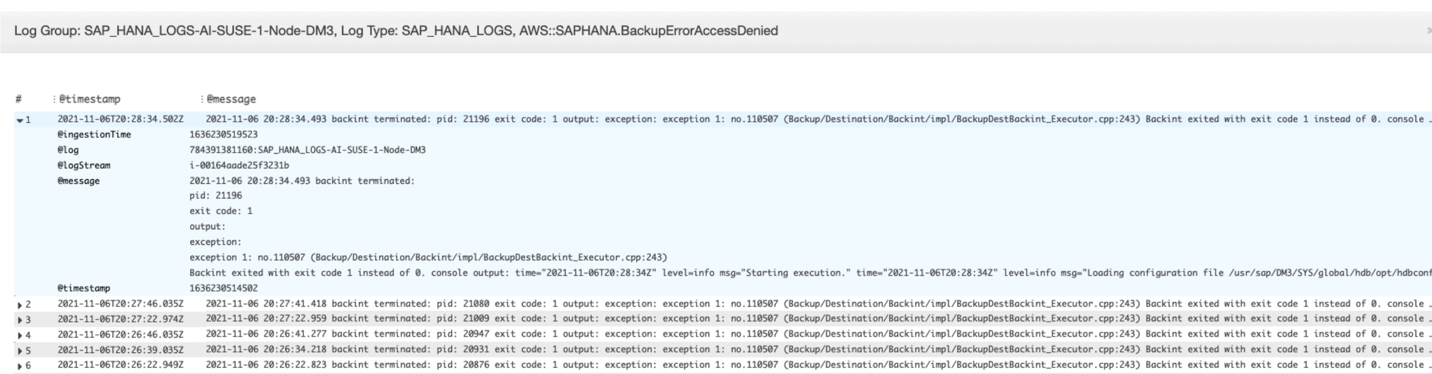
Description

Your SAP application that is backed by an SAP HANA database has stopped working.

Resolution

You can identify the database layer that is causing the problem by checking the dynamically created dashboard, which shows the related metrics and log file snippets.

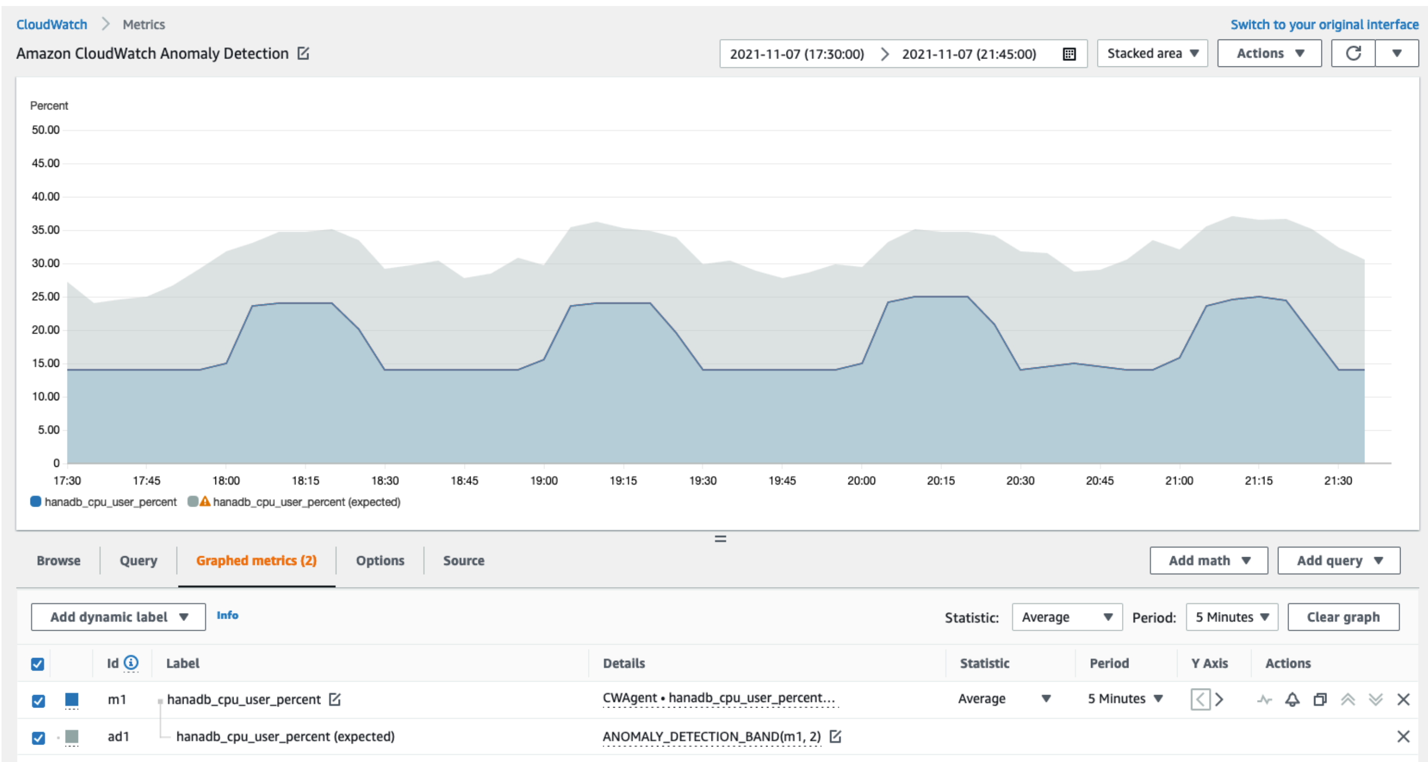
The log group widget in the problem dashboard shows the `ACCESS DENIED` event. This includes additional information, such as the S3 bucket, the S3 bucket folder, and the S3 bucket Region.



Anomaly detection for SAP HANA

For specific SAP HANA metrics, such as the number of thread count, CloudWatch applies statistical and machine learning algorithms to define the threshold. These algorithms continuously analyze the metrics of the SAP HANA database, determine normal baselines, and surface anomalies with minimal user intervention. The algorithms generate an anomaly detection model, which generates a range of expected values that represent normal metric behavior.

Anomaly detection algorithms account for the seasonality and trend changes of metrics. The seasonality changes can be hourly, daily, or weekly, as shown in the following examples of the SAP HANA CPU usage.



After you create a model, CloudWatch anomaly detection continuously evaluates the model and makes adjustments to it to ensure that is it as accurate as possible. This includes retraining the model to adjust if the metric values evolve over time or experience sudden changes. It also includes predictors to improve the models for metrics that are seasonal, spiky, or sparse.

Troubleshooting Application Insights for SAP HANA

This section provides steps to help you resolve common errors returned by the Application Insights dashboard.

Unable to add more than 60 monitored metrics

The output shows the following error.

```
Component cannot have more than 60 monitored metrics
```

Root cause – The current metric limit is 60 monitored metrics per component.

Resolution – To stay under the limit, remove metrics that are not necessary.

No SAP metrics appear after the onboarding process

Use the following information to find out why SAP metrics don't appear on the dashboard after the onboarding process. The first step is to troubleshoot why the SAP metrics don't appear using the Amazon Web Services Management Console or Exporter logs from an Amazon EC2 instance. Next, review the error output to find a resolution.

Troubleshoot why SAP metrics don't appear after onboarding

You can use the Amazon Web Services Management Console or exporter logs from an Amazon EC2 instance for troubleshooting.

Amazon Web Services Management Console

Troubleshoot no SAP metrics appear after onboarding using the console

1. Open the Amazon Systems Manager console at https://console.amazonaws.cn/systems-manager/.
2. In the left navigation pane, choose **State Manager**.
3. Under **Associations**, check the status of the document `AWSEC2-ApplicationInsightsCloudwatchAgentInstallAndConfigure`. If the status is `Failed`, under **Execution id**, select the failed id and view the output.
4. Under **Associations**, check the status of the document `AWS-ConfigureAWSPackage`. If the status is `Failed`, under **Execution id**, select the failed id and view the output.

Exporter logs from Amazon EC2 instance

Troubleshoot no SAP metrics appear after onboarding using exporter logs

1. Connect to the Amazon EC2 instance where your SAP HANA database is running.

2. Find the correct naming convention for WORKLOAD_SHORT_NAME using the following command. You will use this short name in the following two steps.

```
sudo systemctl | grep exporter
```

> **Note**
>
> Application Insights adds a suffix, WORKLOAD_SHORT_NAME to the service name depending on the workload that is running. The short names for SAP HANA single node, multiple nodes, and high availability deployments are HANA_SN, HANA_MN, and HANA_HA.

3. To check for errors in the exporter manager service logs, run the following command replacing WORKLOAD_SHORT_NAME with the short name you found in Step 2.

```
sudo journalctl -e --unit=prometheus-
hanadb_exporter_manager_WORKLOAD_SHORT_NAME.service
```

4. If the exporter manager service logs do not show an error, check for errors in the exporter service logs by running the following command.

```
sudo journalctl -e --unit=prometheus-hanadb_exporter_WORKLOAD_SHORT_NAME.service
```

Resolving the common root causes for SAP metrics not appearing after onboarding

The following examples describe how to resolve the common root causes of SAP metrics not appearing after onboarding.

- The output shows the following error.

```
Reading json config file path: /opt/aws/amazon-cloudwatch-agent/etc/amazon-
cloudwatch-agent.d/default ...
Reading json config file path: /opt/aws/amazon-cloudwatch-agent/etc/
amazon-cloudwatch-agent.d/ssm_AmazonCloudWatch-ApplicationInsights-
SSMParameterForTESTCWEC2INSTANCEi0d88867f1f3e36285.tmp ...
2023/11/30 22:25:17 Failed to merge multiple json config files.
2023/11/30 22:25:17 Failed to merge multiple json config files.
2023/11/30 22:25:17 Under path : /metrics/append_dimensions | Error : Different
 values are specified for append_dimensions
```

```
2023/11/30 22:25:17 Under path : /metrics/metrics_collected/disk | Error : Different
 values are specified for disk
2023/11/30 22:25:17 Under path : /metrics/metrics_collected/mem | Error : Different
 values are specified for mem
2023/11/30 22:25:17 Configuration validation first phase failed. Agent version: 1.0.
 Verify the JSON input is only using features supported by this version.
```

Resolution – Application Insights is trying to configure the same metrics that are pre-configured as part of the existing CloudWatch agent configuration file. Remove the existing files under `/opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.d/` or remove the metrics that are causing the conflict from the existing CloudWatch agent configuration file.

- The output shows the following error.

```
Unable to find a host with system database, for more info rerun using -v
```

  Resolution – The username, password, or database port may be incorrect. Verify that the username, password, and port are valid, then re-run the onboarding process.

- The output shows the following error.

```
This hdbcli installer is not compatible with your Python interpreter
```

  Resolution – Upgrade pip3 and wheel as shown in the following example for Python 3.6.

```
python3.6 -m pip install --upgrade pip setuptools wheel
```

- The output shows the following error.

```
Unable to install hdbcli using pip3. Please try to install it
```

  Resolution – Ensure that you have followed the `hdbclient` prerequisites or install `hdbclient` manually under pip3.

- The output shows the following error.

```
Package 'boto3' requires a different Python: 3.6.15 not in '>= 3.7'
```

  Resolution – Python 3.8 or higher is required for this operating system version. Check the Python 3.8 prerequisites and install it.

- The output shows one of the following installation errors.

```
Can not execute `setup.py` since setuptools is not available in the build environment
```

or

```
[SSL: CERTIFICATE_VERIFY_FAILED]
```

Resolution – Install Python using SUSE Linux commands as shown in the following example. The following example installs the latest version of Python 3.8.

```
wget https://www.python.org/ftp/python/3.8.<LATEST_RELEASE>/
Python-3.8.<LATEST_RELEASE>.tgz
tar xf Python-3.*
cd Python-3.*/
sudo zypper install make gcc-c++ gcc automake autoconf libtool
sudo zypper install zlib-devel
sudo zypper install libopenssl-devel libffi-devel
./configure --with-ensurepip=install
sudo make
sudo make install
sudo su
python3.8 -m pip install --upgrade pip setuptools wheel
```

Tutorial: Set up monitoring for SAP NetWeaver

This tutorial demonstrates how to configure Amazon CloudWatch Application Insights to set up monitoring for SAP NetWeaver. You can use CloudWatch Application Insights automatic dashboards to visualize problem details, accelerate troubleshooting, and reduce mean time to resolution (MTTR) for your SAP NetWeaver application servers.

CloudWatch Application Insights for SAP NetWeaver topics

- Supported environments
- Supported operating systems
- Features
- Prerequisites

- Set up your SAP NetWeaver application servers for monitoring
- Manage monitoring of your SAP NetWeaver application servers
- View and troubleshoot SAP NetWeaver problems detected by CloudWatch Application Insights
- Troubleshooting Application Insights for SAP NetWeaver

Supported environments

CloudWatch Application Insights supports the deployment of Amazon resources for the following systems and patterns.

- **SAP NetWeaver Standard System Deployment.**
- **SAP NetWeaver Distributed deployments on multiple Amazon EC2 instances.**
- **Cross-AZ SAP NetWeaver high availability setup** – SAP NetWeaver with high availability configured across two Availability Zones using SUSE/RHEL clustering.

Supported operating systems

CloudWatch Application Insights for SAP NetWeaver is supported on the following operating systems:

- Oracle Linux 8
- Red Hat Enterprise Linux 7.6
- Red Hat Enterprise Linux 7.7
- Red Hat Enterprise Linux 7.9
- Red Hat Enterprise Linux 8.1
- Red Hat Enterprise Linux 8.2
- Red Hat Enterprise Linux 8.4
- Red Hat Enterprise Linux 8.6
- SUSE Linux Enterprise Server 15 for SAP
- SUSE Linux Enterprise Server 15 SP1 for SAP
- SUSE Linux Enterprise Server 15 SP2 for SAP
- SUSE Linux Enterprise Server 15 SP3 for SAP
- SUSE Linux Enterprise Server 15 SP4 for SAP

- SUSE Linux Enterprise Server 12 SP4 for SAP
- SUSE Linux Enterprise Server 12 SP5 for SAP
- SUSE Linux Enterprise Server 15 except High Availability patterns
- SUSE Linux Enterprise Server 15 SP1 except High Availability patterns
- SUSE Linux Enterprise Server 15 SP2 except High Availability patterns
- SUSE Linux Enterprise Server 15 SP3 except High Availability patterns
- SUSE Linux Enterprise Server 15 SP4 except High Availability patterns
- SUSE Linux Enterprise Server 12 SP4 except High Availability patterns
- SUSE Linux Enterprise Server 12 SP5 except High Availability patterns

Features

CloudWatch Application Insights for SAP NetWeaver 7.0x–7.5x (including ABAP Platform) provides the following features:

- Automatic SAP NetWeaver workload detection
- Automatic SAP NetWeaver alarm creation based on static thresholds
- Automatic SAP NetWeaver log pattern recognition
- Health dashboard for SAP NetWeaver
- Problem dashboard for SAP NetWeaver

Prerequisites

You must perform the following prerequisites to configure SAP NetWeaver with CloudWatch Application Insights:

- **Amazon Systems Manager enablement** – Install SSM Agent on your Amazon EC2 instances, and enable the instances for SSM. For information about how to install the SSM Agent, see [Setting up Amazon Systems Manager](#) in the *Amazon Systems Manager User Guide*.
- **Amazon EC2 instance roles** – You must attach the following Amazon EC2 instance roles to configure your SAP NetWeaver monitoring.
  - You must attach the `AmazonSSMManagedInstanceCore` role to enable Systems Manager. For more information, see [Amazon Systems Manager identity-based policy examples](#).

  - You must attach the `CloudWatchAgentServerPolicy` policy to enable instance metrics and logs to be emitted through CloudWatch. For more information, see Create IAM roles and users for use with CloudWatch agent.

- **Amazon resource groups –** You must create a resource group that includes all of the associated Amazon resources used by your application stack to onboard your applications to CloudWatch Application Insights. This includes Amazon EC2 instances, Amazon EFS, and Amazon EBS volumes running your SAP NetWeaver application servers. If there are multiple SAP NetWeaver systems per account, we recommend that you create one resource group that includes the Amazon resources for each SAP NetWeaver system. For more information about creating resource groups, see the *Amazon Resource Groups and Tags User Guide*.
- **IAM permissions –** For users who don't have administrative access, you must create an Amazon Identity and Access Management (IAM) policy that allows Application Insights to create a service-linked role and attach it to the user's identity. For more information about how to create the IAM policy, see IAM policy.
- **Service-linked role –** Application Insights uses Amazon Identity and Access Management (IAM) service-linked roles. A service-linked role is created for you when you create your first Application Insights application in the Application Insights console. For more information, see Using service-linked roles for CloudWatch Application Insights.
- **Amazon CloudWatch agent –** Application Insights installs and configures the CloudWatch agent. If you have CloudWatch agent installed, Application Insights retains your configuration. To avoid a merge conflict, remove the configuration of resources that you want to use in Application Insights from the existing CloudWatch agent configuration file. For more information, see Manually create or edit the CloudWatch agent configuration file.

Set up your SAP NetWeaver application servers for monitoring

Use the following steps to set up monitoring for your SAP NetWeaver application servers.

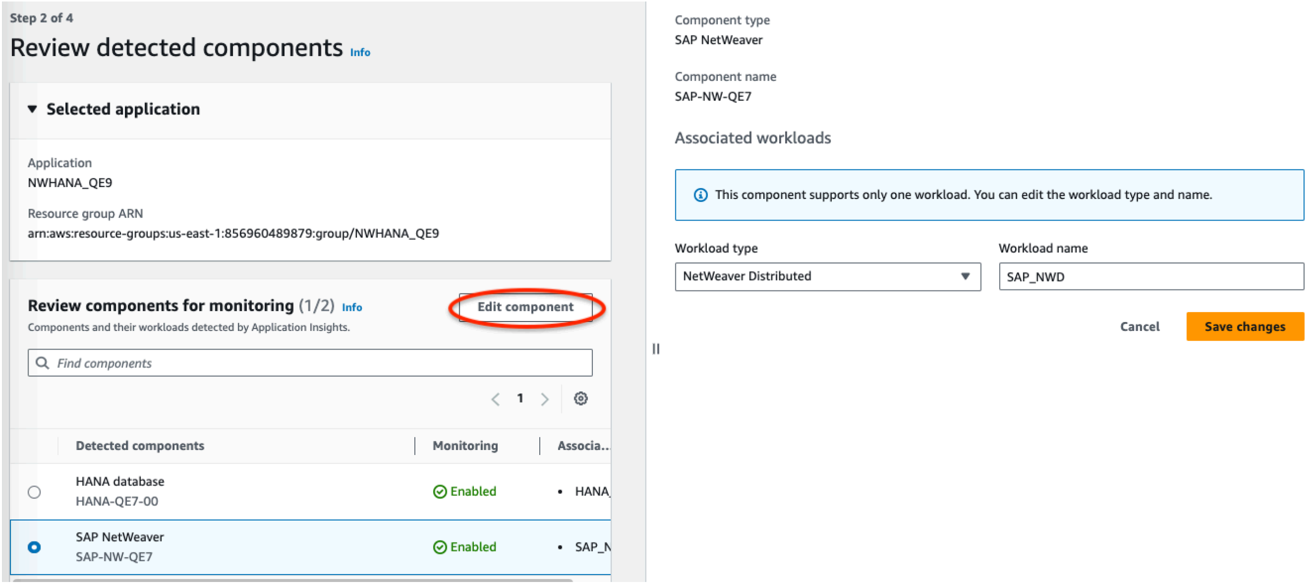
To set up monitoring

1. Open the CloudWatch console.
2. From the left navigation pane, under **Insights**, select **Application Insights**.
3. The **Application Insights** page displays the list of applications that are monitored with Application Insights, and the monitoring status for each application. In the upper right-hand corner, select **Add an application**.

4. On the **Specify application details** page, from the dropdown list under **Resource group**, select the Amazon resource group you created that contains your SAP NetWeaver resources. If you haven't created a resource group for your application, you can create one by choosing **Create new resource group** under the **Resource group** dropdown list.
5. Under **Automatic monitoring of new resources**, select the check box to allow Application Insights to automatically monitor the resources that are added to the application's resource group after onboarding.
6. Under **Monitor EventBridge events**, select the check box to integrate Application Insights monitoring with CloudWatch Events to get insights from Amazon EBS, Amazon EC2, Amazon CodeDeploy, Amazon ECS, Amazon Health APIs and notifications, Amazon RDS, Amazon S3, and Amazon Step Functions.
7. Under **Integrate with Amazon Systems Manager OpsCenter**, select the check box next to **Generate Amazon Systems Manager OpsCenter OpsItems for remedial actions** to view and get notifications when problems are detected for the selected applications. To track the operations that are performed to resolve operational work items, called OpsItems, that are related to your Amazon resources, provide an SNS topic ARN.
8. You can optionally enter tags to help you identify and organize your resources. CloudWatch Application Insights supports both tag-based and Amazon CloudFormation stack-based resource groups, with the exception of Application Auto Scaling groups. For more information, see Tag Editor in the *Amazon Resource Groups and Tags User Guide*.
9. To review detected components, choose **Next**.
10. On the **Review detected components** page, the monitored components and their workloads automatically detected by CloudWatch Application Insights are listed.

    - To edit the workload type and name, choose **Edit component**.

      > **Note**
      >
      > Components that contain a detected NetWeaver Distributed or NetWeaver High Availability workload support only one workload on a component.



11. Choose **Next**.
12. On the **Specify component details** page, choose **Next**.
13. Review your application monitoring configuration, then choose **Submit**.
14. The application details page opens, where you can view the **Application summary**, **Dashboard**, **Components**, and **Workloads**. You can also view the **Configuration history**, **Log patterns**, and any **Tags** that you have created. After you submit your application, CloudWatch Application Insights deploys all of the metrics and alarms for your SAP NetWeaver system, which can take up to an hour.

Manage monitoring of your SAP NetWeaver application servers

Use the following steps to manage monitoring of your SAP NetWeaver application servers.

To manage monitoring

1. Open the CloudWatch console.
2. From the left navigation pane, under **Insights**, select **Application Insights**.
3. Choose the **List view** tab.
4. The **Application Insights** page displays the list of applications that are monitored with Application Insights, and the monitoring status for each application.
5. Select your application.

6. Choose the **Components** tab.
7. Under **Monitored components**, select the radio button next to the component name. Then, select **Manage monitoring**.
8. Under **Instance logs**, you can update the existing log path, log pattern set, and log group name. In addition, you can add up to three additional **Application logs**.
9. Under **Metrics**, you can select the SAP NetWeaver metrics according to your requirements. SAP NetWeaver metric names are prefixed with `sap`. You can add up to 40 metrics per component.
10. Under **Custom alarms**, you can add additional alarms to be monitored by CloudWatch Application Insights.
11. Review your application monitoring configuration and choose **Save**. When you submit your configuration, your account updates all of the metrics and alarms for your SAP NetWeaver systems.

View and troubleshoot SAP NetWeaver problems detected by CloudWatch Application Insights

The following sections provide steps to help you resolve common troubleshooting scenarios that occur when you configure monitoring for SAP NetWeaver on Application Insights.

Troubleshooting topics

- SAP NetWeaver database connectivity issues
- SAP NetWeaver application availability issues

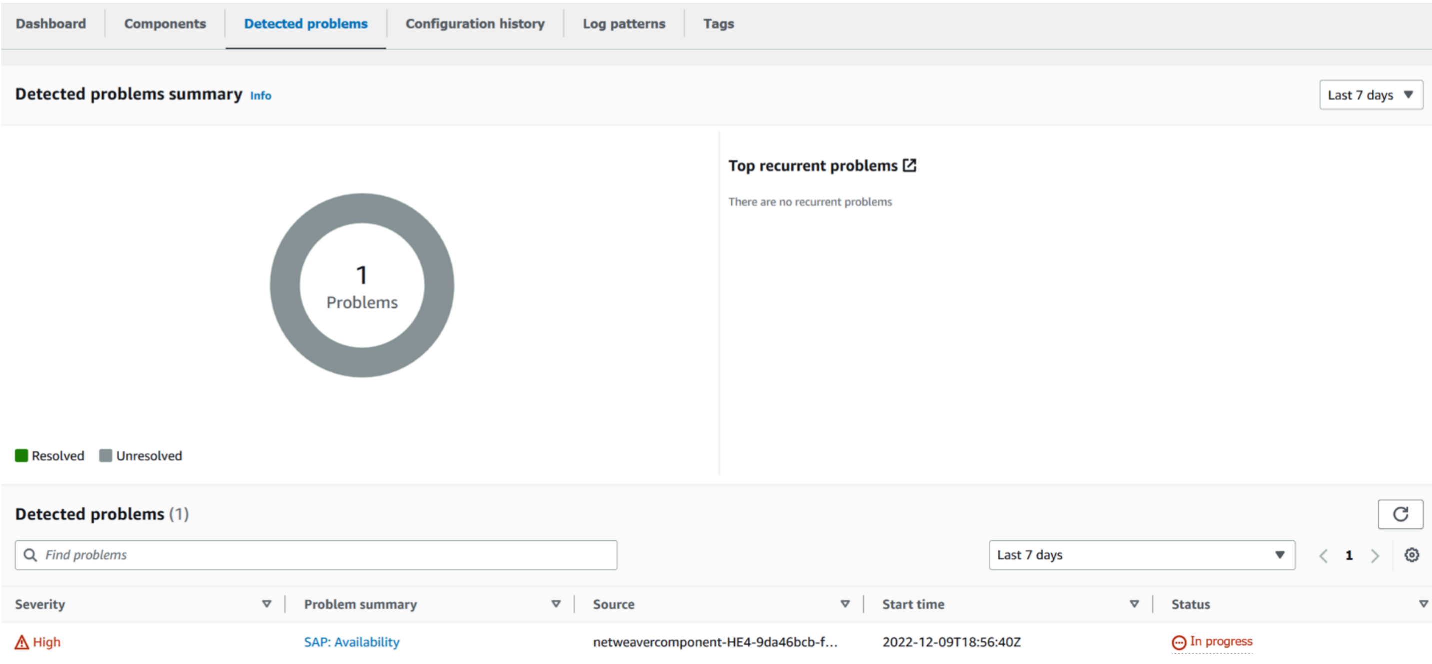
SAP NetWeaver database connectivity issues

Description

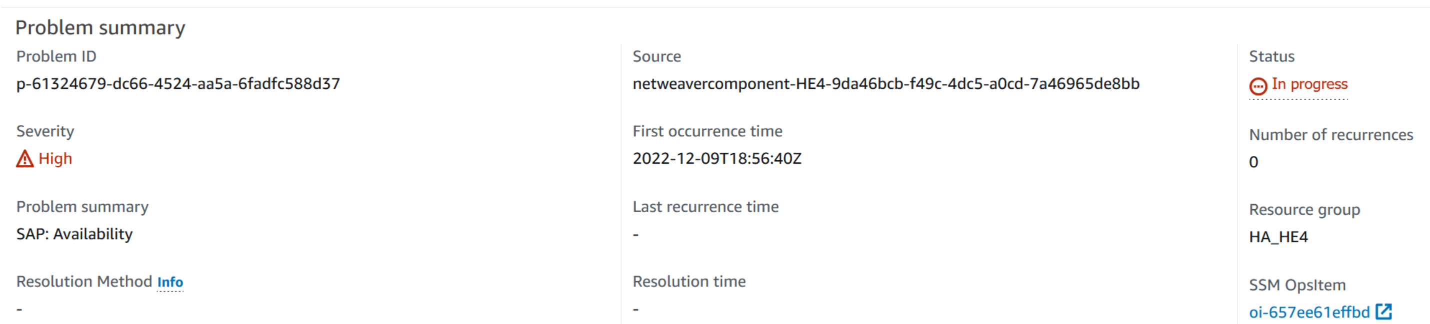
Your SAP NetWeaver application experiences database connectivity issues.

Cause

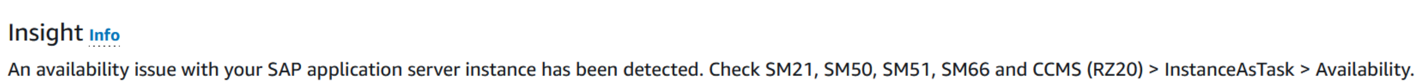
You can identify the connectivity issue by going to the CloudWatch Application Insights console and checking the SAP NetWeaver Application Insights problem dashboard. Select the link under **Problem summary** to see the specific issue.



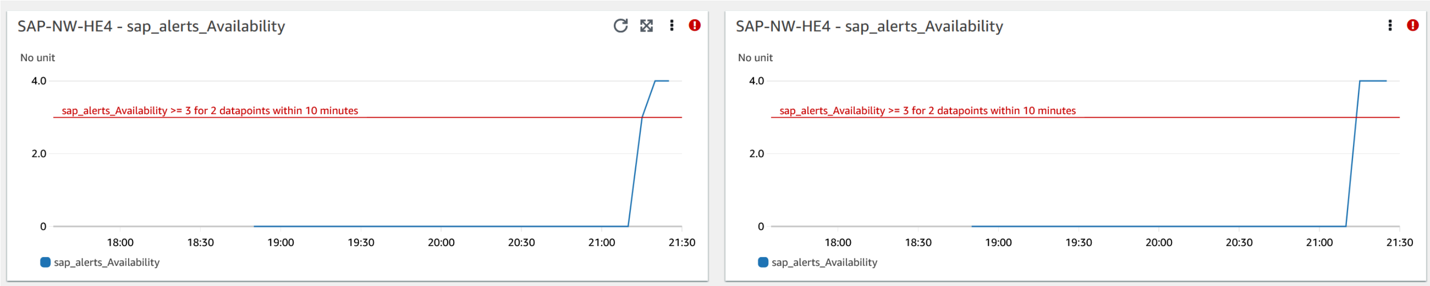
In the following example, under **Problem summary**, SAP: Availability is the issue.



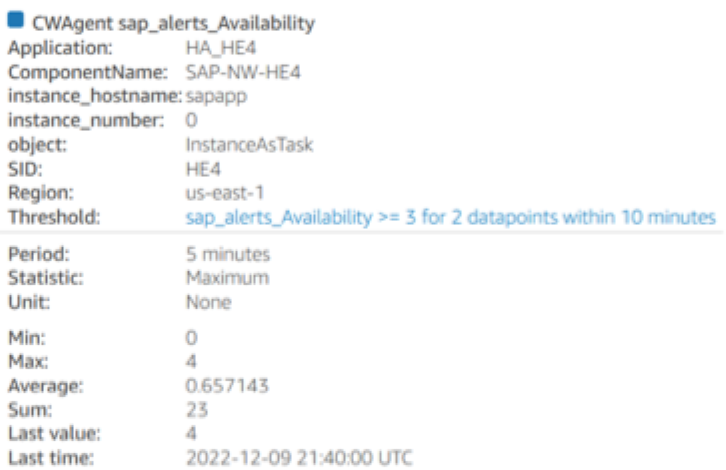
Immediately following the **Problem summary**, the **Insight** section provides more context about the error and where you can get more information about the causes of the issue.



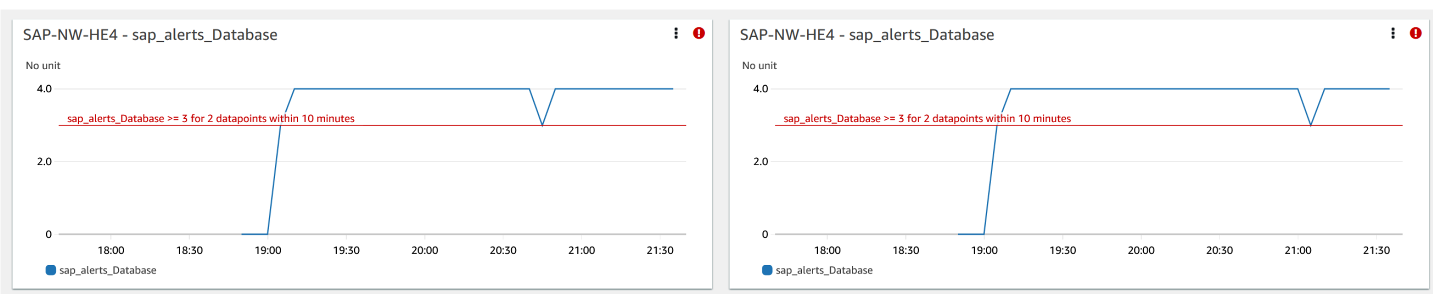
On the same problem dashboard, you can view related logs and metrics that problem detection has grouped together to help you isolate the cause of the error. The `sap_alerts_Availability` metric tracks the availability of the SAP NetWeaver system over time. You can use historical tracking to correlate when the metric initiated an error state or breached the alarm threshold. In the following example, there is an availability issue with the SAP NetWeaver system. The example shows two alarms because there are two SAP application server instances and an alarm was created for each instance.



For more information about each alarm, hover over the `sap_alerts_Availability` metric name.



In the following example, the `sap_alerts_Database` metric shows that the database layer has an issue or a failure. This alarm indicates that SAP NetWeaver had issues connecting to or communicating with its database.

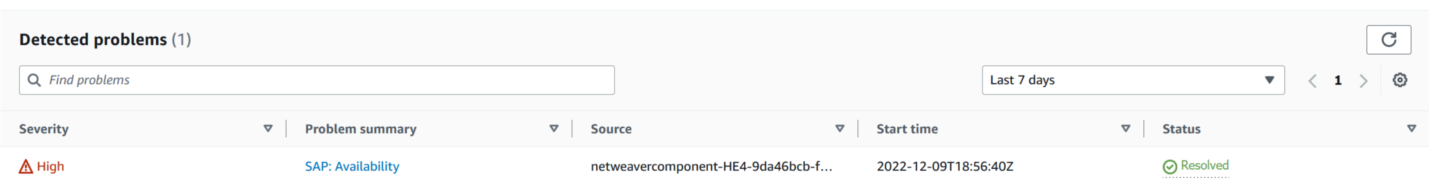


Since the database is a key resource for SAP NetWeaver, you may get many related alarms when the database has an issue or failure. In the following example, the `sap_alerts_FrontendResponseTime` and `sap_alerts_LongRunners` metrics are initiated because the database is not available.

Resolution

Application Insights monitors the detected problem hourly. If there are no new related log entries in your SAP NetWeaver log files, the older log entries will be treated as resolved. You must fix any error conditions related to the CloudWatch alarms. After the error conditions are fixed, the alarm is resolved when the alarms and logs are recovered. When all of the CloudWatch log errors and alarms are resolved, Application Insights stops detecting errors and the problem is automatically resolved within an hour. We recommend that you resolve all log error conditions and alarms so that you have the latest problems on the problem dashboard.

In the following example, the SAP Availability issue is resolved.



Detected problems (1)

Find problems | Last 7 days

Severity	Problem summary	Source	Start time	Status
High	SAP: Availability	netweavercomponent-HE4-9da46bcb-f...	2022-12-09T18:56:40Z	Resolved

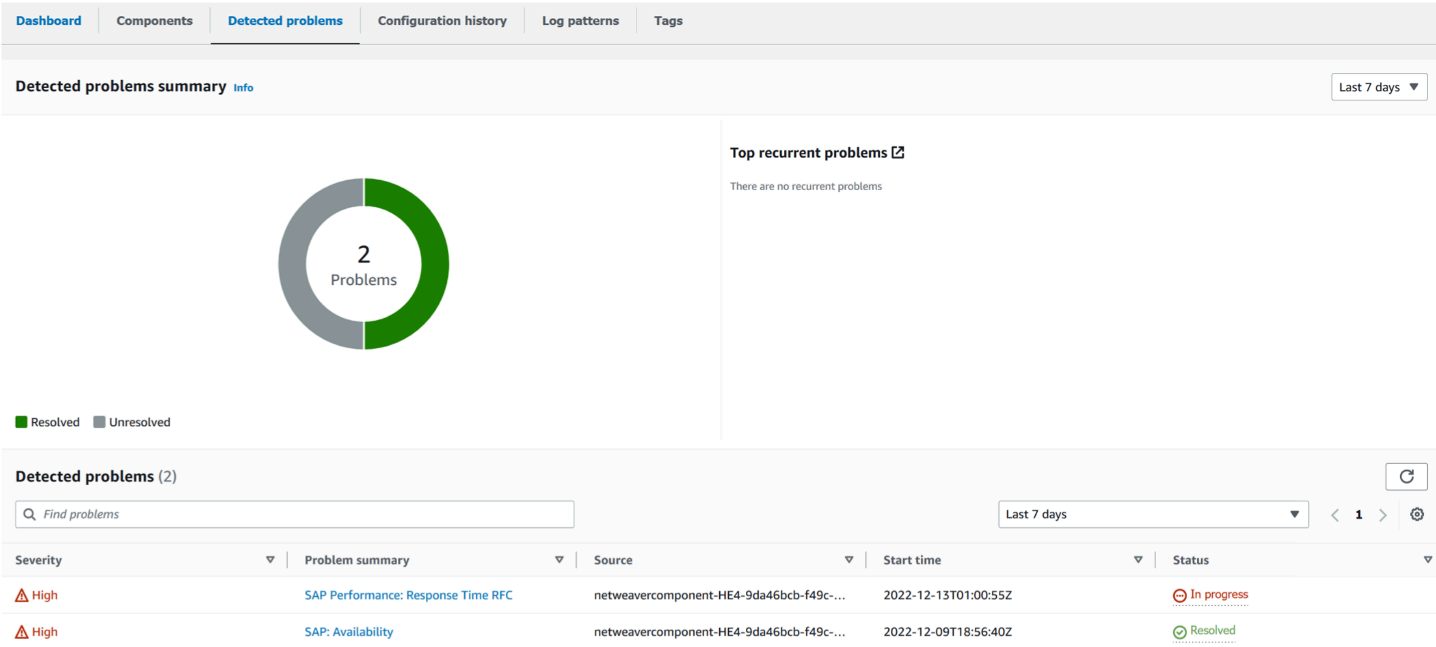
SAP NetWeaver application availability issues

Description

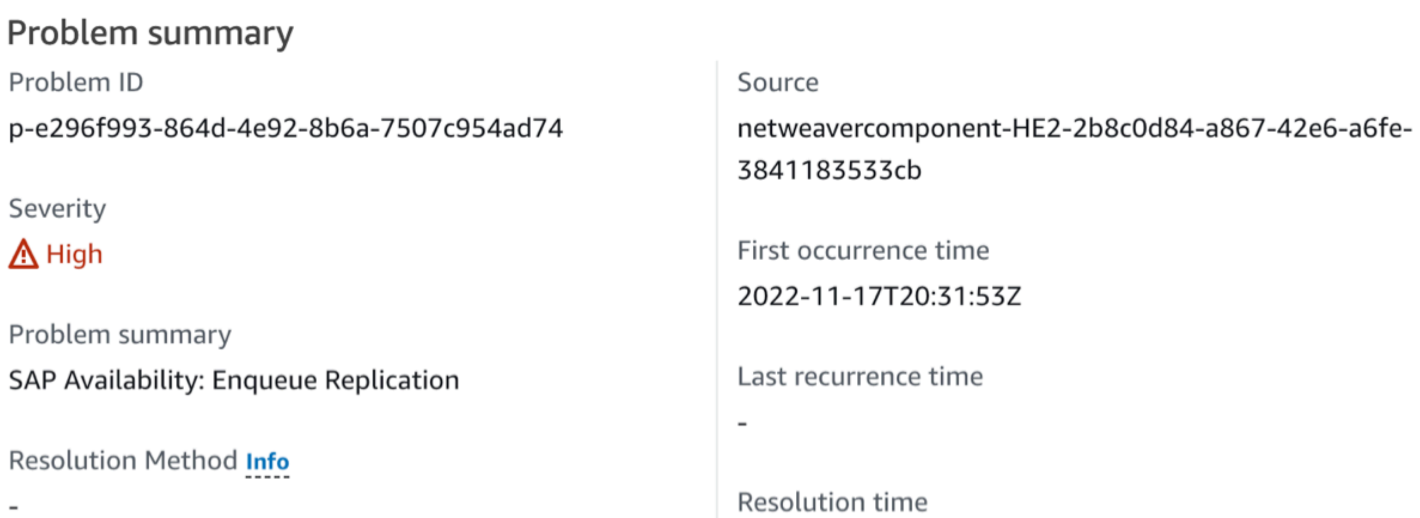
Your SAP NetWeaver High Availability Enqueue replication stopped working.

Cause

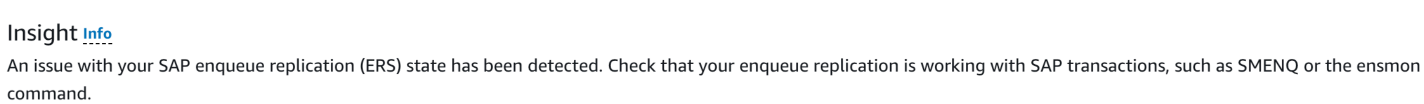
You can identify the connectivity issue by going to the CloudWatch Application Insights console and checking the SAP NetWeaver Application Insights problem dashboard. Select the link under **Problem summary** to see the specific issue.



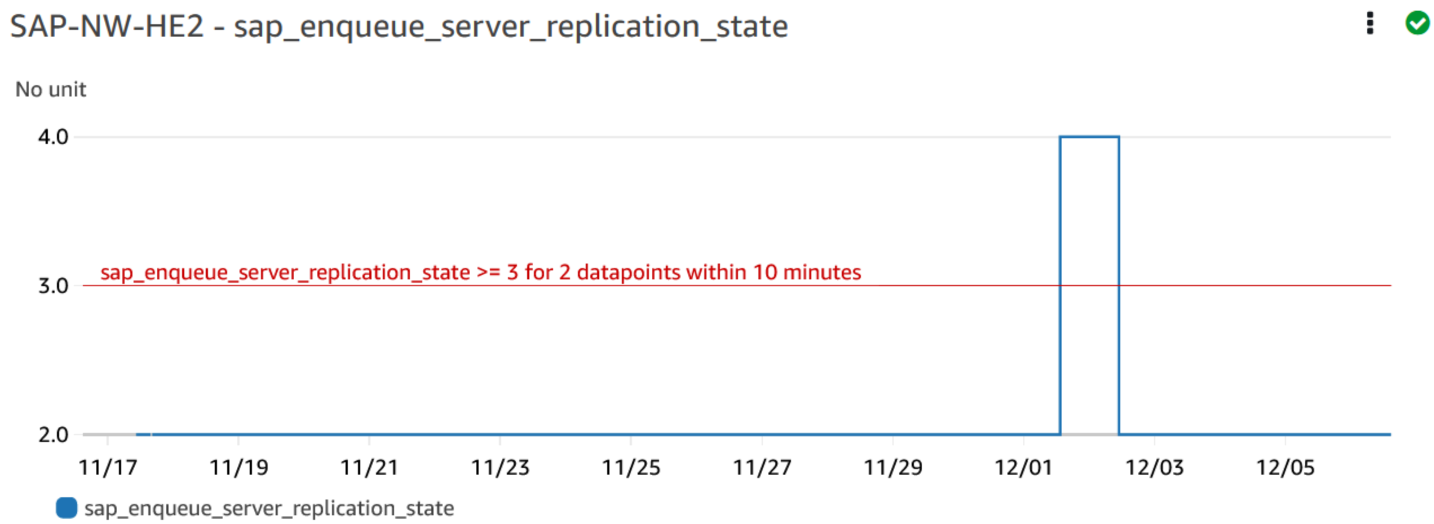
In the following example, under **Problem summary**, High Availability Enqueue Replication is the issue.



Immediately following the **Problem summary**, the **Insight** section provides more context about the error and where you can get more information about the causes of the issue.

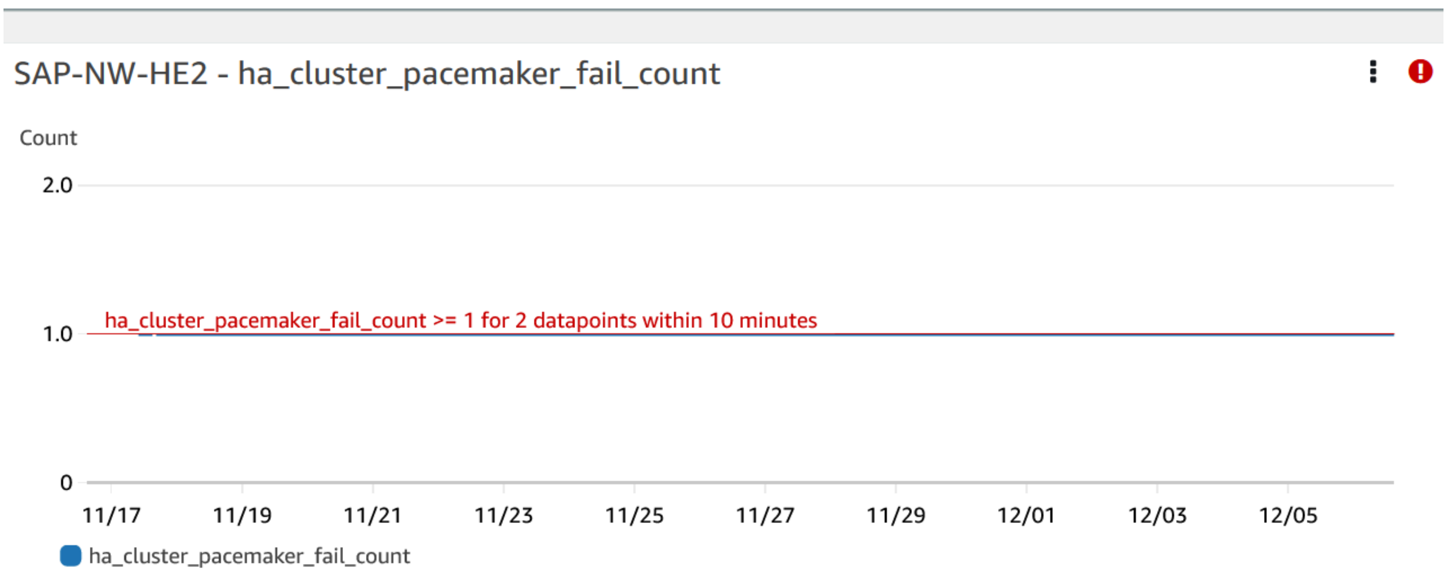


The following example shows the problem dashboard where you view logs and metrics which are grouped to help you isolate the causes of the error. The `sap_enqueue_server_replication_state` metric tracks the value over time. You can use historical tracking to correlate when the metric initiated an error state or breached the alarm threshold.

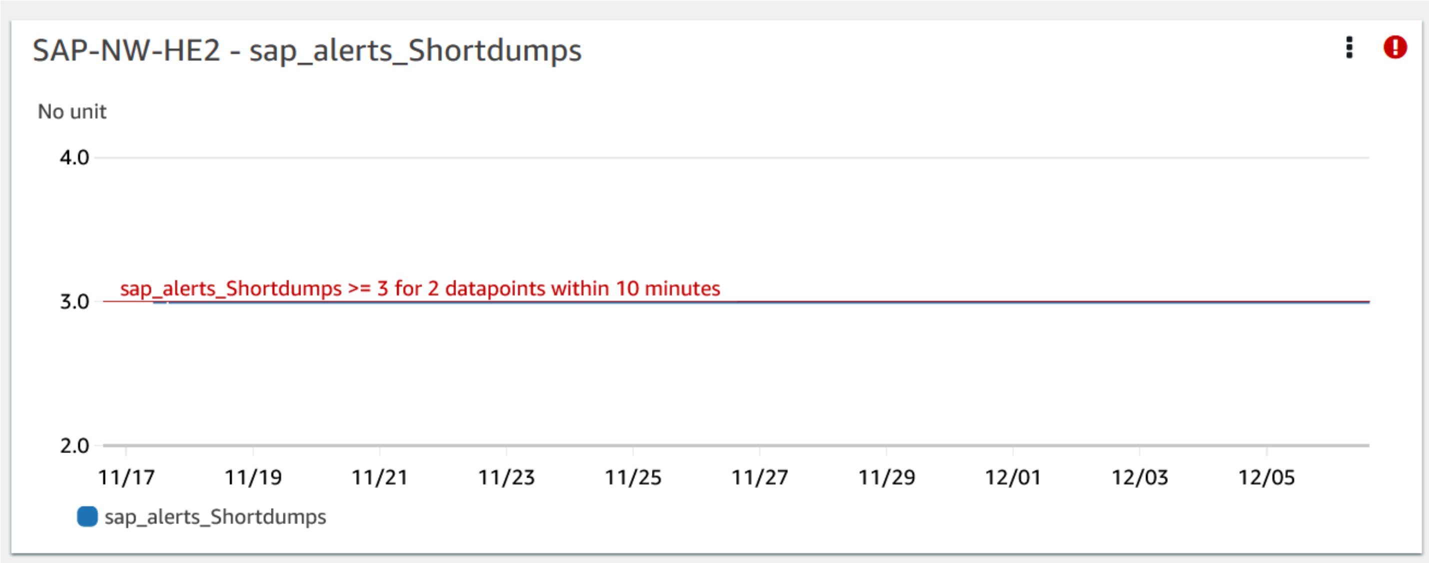


In the following example, the `ha_cluster_pacemaker_fail_count` metric shows that the high availability pacemaker cluster experienced a resource failure. The specific pacemaker resources that had a fail count greater than or equal to one are identified in the component dashboard.

EC2 instance group - SAP-NW-HE2

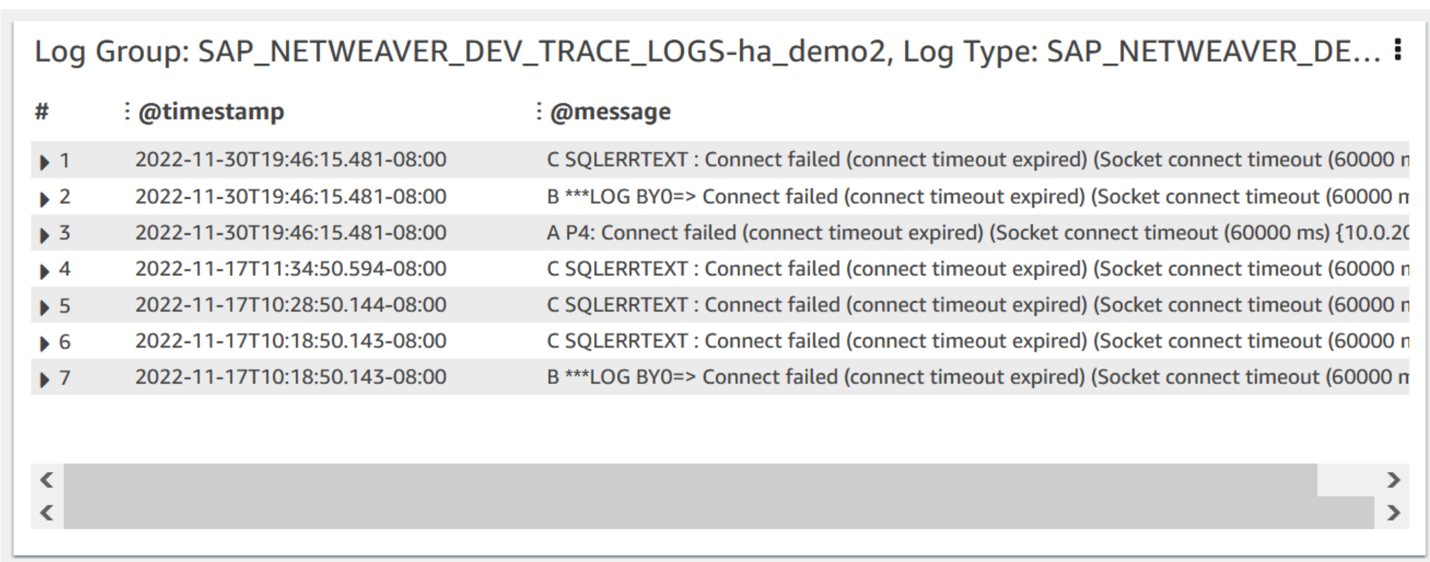


The following example shows the `sap_alerts_Shortdumps` metric, which indicates that the SAP application performance was reduced when the problem was detected.



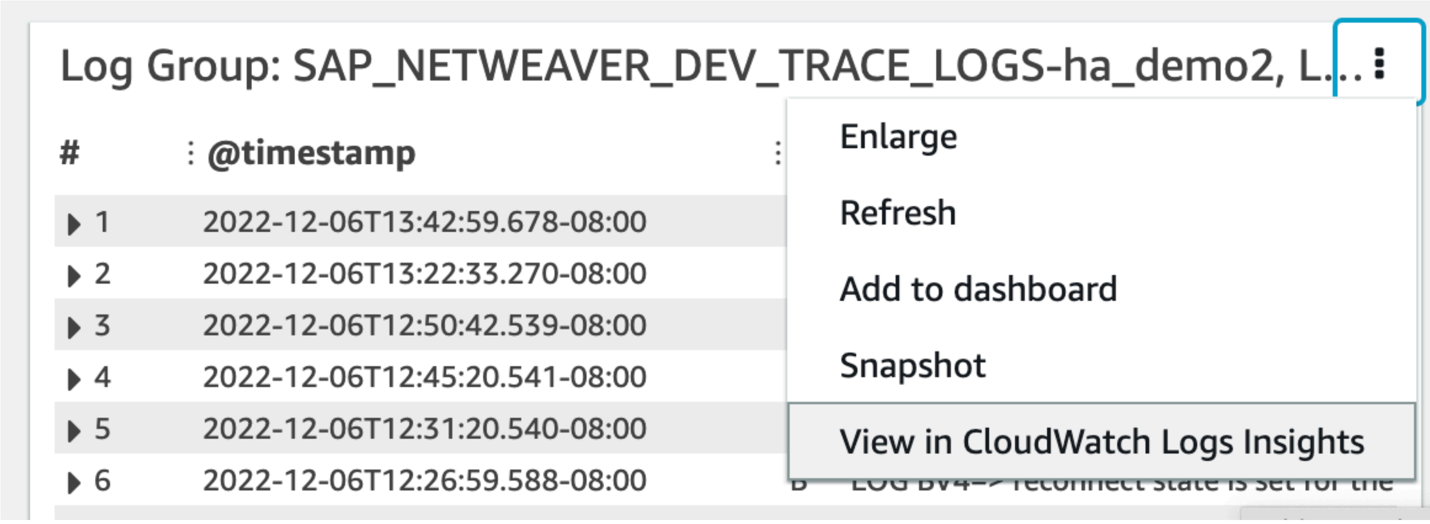
Logs

The log entries are helpful to get a better understanding of issues that occurred at the SAP NetWeaver layer when the problem was detected. The log group widget in the problem dashboard shows the specific time of the issue.


Log Group: SAP_NETWEAVER_DEV_TRACE_LOGS-ha_demo2, Log Type: SAP_NETWEAVER_DE...

#	@timestamp	@message
1	2022-11-30T19:46:15.481-08:00	C SQLERRTEXT : Connect failed (connect timeout expired) (Socket connect timeout (60000 n
2	2022-11-30T19:46:15.481-08:00	B ***LOG BY0=> Connect failed (connect timeout expired) (Socket connect timeout (60000 n
3	2022-11-30T19:46:15.481-08:00	A P4: Connect failed (connect timeout expired) (Socket connect timeout (60000 ms) {10.0.2(
4	2022-11-17T11:34:50.594-08:00	C SQLERRTEXT : Connect failed (connect timeout expired) (Socket connect timeout (60000 n
5	2022-11-17T10:28:50.144-08:00	C SQLERRTEXT : Connect failed (connect timeout expired) (Socket connect timeout (60000 n
6	2022-11-17T10:18:50.143-08:00	C SQLERRTEXT : Connect failed (connect timeout expired) (Socket connect timeout (60000 n
7	2022-11-17T10:18:50.143-08:00	B ***LOG BY0=> Connect failed (connect timeout expired) (Socket connect timeout (60000 n

To see detailed information about the logs, select the three vertical dots in the upper-right corner, and select **View in CloudWatch Logs Insights**.

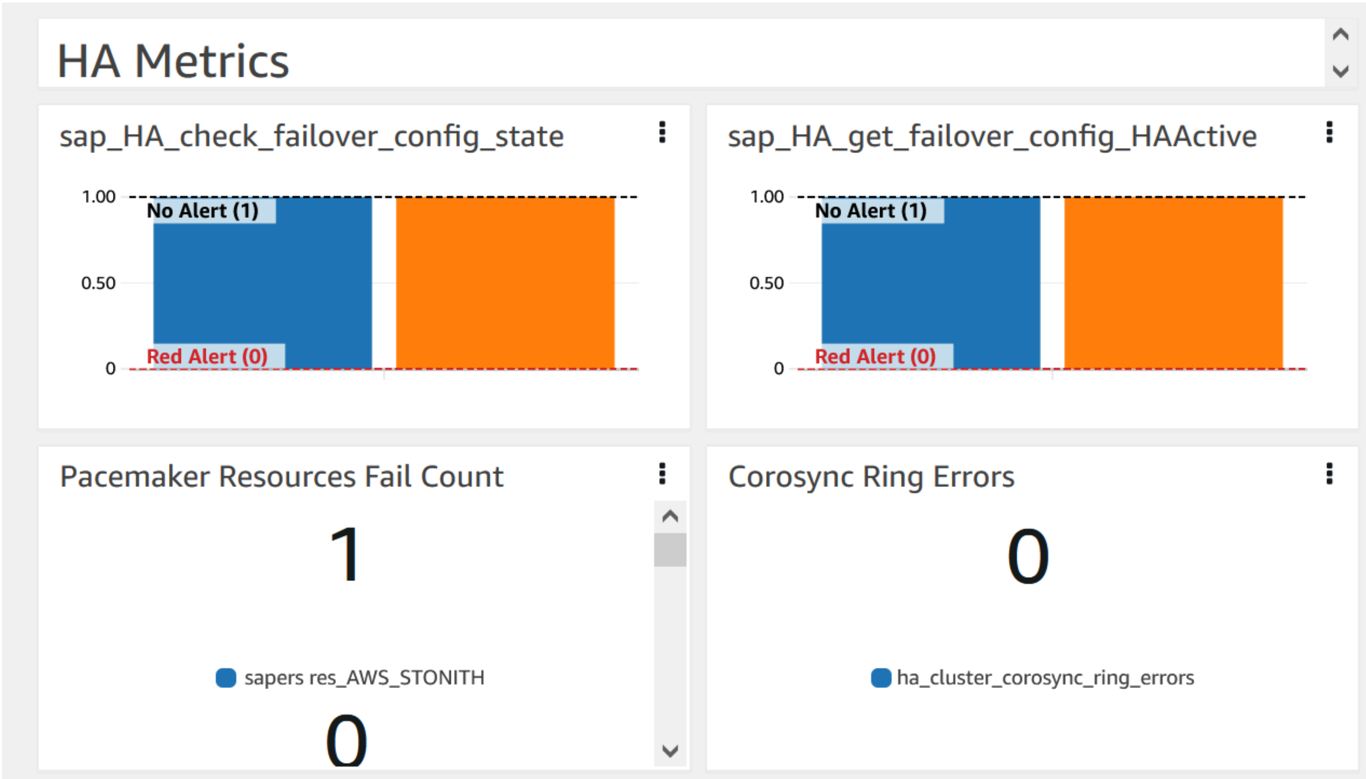


Use the following steps to get more information about the metrics and alarms displayed in the problem dashboard.

To get more information about metrics and alarms

1. Open the [CloudWatch console](https://console.aws.amazon.com/cloudwatch/).
2. In the left navigation pane, under **Insights**, select **Application Insights**. Then, choose the **List view** tab, and select your application.
3. Select the **Components** tab. Then, select the SAP NetWeaver component about which you want to get more information.

   The following example shows the **HA Metrics** section with the `ha_cluster_pacemaker_fail_count` metric that was displayed in the problem dashboard.



Resolution

Application Insights monitors the detected problem hourly. If there are no new related log entries in your SAP NetWeaver log files, the older log entries will be treated as resolved. You must fix any error conditions related to this problem.

For the `sap_alerts_Shortdumps` alarm, you must resolve the alert in the SAP NetWeaver system by using transaction code `RZ20 # R3Abap # Shortdumps` to navigate to the CCMS alert. For more information about CCMS alerts, see the [SAP website.](#) Resolve all of the CCMS alerts in the Shortdumps tree. After all of the alerts are resolved in the SAP NetWeaver system, CloudWatch no longer reports the metric in an alarm state.

When all of the CloudWatch log errors and alarms are resolved, Application Insights stops detecting errors and the problem is automatically resolved within an hour. We recommend that you resolve all log error conditions and alarms so that you have the latest problems on the problem dashboard. In the following example, the SAP Netweaver High Availability Enqueue Replication problem is resolved.

Severity	Problem summary	Source	Start time	Status
High	SAP Availability: Enqueue Replication	netweavercomponent-HE2-2b8c0...	2022-12-08T20:01:43Z	Resolved

Troubleshooting Application Insights for SAP NetWeaver

This section provides steps to help you resolve common errors returned by the Application Insights dashboard.

Unable to add more than 60 monitor metrics

Error returned: `Component cannot have more than 60 monitored metrics.`

Root cause: `The current metric limit is 60 monitor metrics per component.`

Resolution: Remove metrics that are not necessary to adhere to the limit.

SAP metrics do not appear on the dashboard after the onboarding process

Root cause: Component Dashboard uses a five minute metric period to aggregate the data points.

Resolution: All metrics should show up on the dashboard after five minutes.

SAP metrics and alarms don't appear on the dashboard

Use the following steps to identify why SAP metrics and alarms don't appear on the dashboard after the onboarding process.

To identify the issue with metrics and alarms

1. Open the CloudWatch console.
2. In the left navigation pane, under **Insights**, select **Application Insights**. Then, choose the **List view** tab, and select your application.
3. Choose the **Configuration history** tab.
4. If you see missing metrics datapoints, check for errors related to `prometheus-sap_host_exporter`.
5. If you don't find an error in the previous step, Connect to your Linux instance. For High Availability deployments, connect to the primary cluster Amazon EC2 instance.
6. In your instance, verify that the exporter is running by using the following command. The default port is `9680`. If you are using a different port, replace `9680` with the port you are using.

   ```
   curl localhost:9680/metrics
   ```

   If no data is returned, then the exporter failed to start.

7. To find the correct naming convention to use for WORKLOAD_SHORT_NAME in the next two steps, run the following command.

   > **Note**
   >
   > Application Insights adds a suffix, WORKLOAD_SHORT_NAME, to the service name depending on the workload that is running. The short names for NetWeaver Distributed, Standard, and High Availability deployments are SAP_NWD, SAP_NWS, and SAP_NWH.

   ```
   sudo systemctl | grep exporter
   ```

8. To check for errors in the exporter service logs, run the following command:

   ```
   sudo journalctl -e --unit=prometheus-sap_host_exporter_WORKLOAD_SHORT_NAME.service
   ```

9. To check for errors in the exporter manager service logs, run the following command:

   ```
   sudo journalctl -e --unit=prometheus-
   sap_host_exporter_manager_WORKLOAD_SHORT_NAME.service
   ```

   > **Note**
   >
   > This service should be up and running at all times.

   If this command does not return an error, continue to the next step.

10. To manually start the exporter, run the following command. Then, check the exporter output.

   ```
   sudo /opt/aws/sap_host_exporter/sap_host_exporter
   ```

   You can exit the exporter process after you check for errors.

Root cause: There are several possible causes for this issue. A common cause is that the exporter is not able to connect to one of the application server instances.

Resolution

Use the following steps to connect the exporter to the application server instances. You will verify that the SAP application instance is running and use SAPControl to connect to the instance.

To connect the exporter to the application server instances

1. In your Amazon EC2 instance, run the following command to verify that the SAP application is running.

   ```
   sapcontrol -nr <App_InstNo> -function GetProcessList
   ```

2. You must establish a working SAPControl connection. If the SAPControl connection doesn't work, find the root cause of the issue on the relevant SAP application instance.
3. To manually start the exporter after you fix the SAP Control connection issue, run the following command:

   ```
   sudo systemctl start prometheus-sap_host_exporter.service
   ```

4. If you can't resolve the SAPControl connection issue, use the following procedure as a temporary fix.

   a. Open the Amazon Systems Manager console.

   b. From the left navigation pane, choose **State Manager**.

   c. Under **Associations** search for the SAP NetWeaver system's association.

      ```
      Association Name: Equal: AWS-ApplicationInsights-
      SSMSAPHostExporterAssociationForCUSTOMSAPNW<SID>-1
      ```

   d. Select the **Association id**.

   e. Choose the **Parameters** tab and remove the application server number from **additionalArguments**.

   f. Choose **Apply Association Now**.

      Note

      This is a temporary fix. If updates are made to the component's monitoring configurations, the instance will be added back.

View and troubleshoot problems detected by Amazon CloudWatch Application Insights

The topics in this section provide detailed information about the detected problems and insights displayed by Application Insights. It also provides suggested resolutions for detected issues with your account or your configuration.

Troubleshooting topics

CloudWatch console overview

An overview of problems that impact your monitored applications can be found under the CloudWatch Application Insights pane in the overview page of the CloudWatch console.

The CloudWatch Application Insights overview pane displays the following:

- The severity of the problems detected: High/Medium/Low
- A short summary of the problem
- The problem source
- The time the problem started
- The resolution status of the problem
- The affected resource group

To view the details of a specific problem, under **Problem Summary**, select the description of the problem. A detailed dashboard displays insights into the problem and related metric anomalies and snippets of log errors. You can provide feedback on the relevance of the insight by selecting whether it is useful.

If a new resource is detected that is not configured, the problem summary description takes you to the **Edit configuration** wizard to configure your new resource. You can view or edit your resource group configuration by choosing **View/edit configuration** in the upper right-hand corner of the detailed dashboard.

To return to the overview, choose **Back to overview**, which is next to the CloudWatch Application Insights detailed dashboard header.

Application Insights problem summary page

Application Insights problem summary page

CloudWatch Application Insights provides the following information about detected problems on the problem summary page:

- A short summary of the problem
- The start time and date of the problem
- The problem severity: High/Medium/Low
- The status of the detected problem: In-progress/Resolved
- Insights: Automatically generated insights on the detected problem and possible root cause
- Feedback on insights: Feedback you have provided about the usefulness of the insights generated by CloudWatch Application Insights
- Related observations: A detailed view of the metric anomalies and error snippets of relevant logs related to the problem across various application components

CloudWatch agent merge conflict failures

CloudWatch Application Insights installs and configures the CloudWatch agent on customer instances. This includes creation of a CloudWatch agent configuration file with configurations for metrics or logs. A merge conflict can occur if a customer's instance already has a CloudWatch agent configuration file with different configurations defined for the same metrics or logs. To resolve the merge conflict, use the following steps:

1. Identify the CloudWatch agent configuration files on your system. For more information about the file locations, see [CloudWatch agent files and locations]().

2. Remove the resource configurations that you want to use in Application Insights from the existing CloudWatch agent configuration file. If you want to only use Application Insights configurations, delete the existing CloudWatch agent configuration files.

High CPU usage from CloudWatch agent log processing

CloudWatch Application Insights installs and configures the CloudWatch agent on customer instances. If an Amazon EC2 instance is configured with log paths that have large amounts of log data, the instance might experience increases in CPU usage while the CloudWatch agent processes the logs. To reduce CPU usage, remove the log path in the Amazon EC2 instance component configuration.

Alarms are not created

For some metrics, Application Insights predicts the alarm threshold based on previous data points for the metric. To enable this prediction, the following criteria must be met.

- **Recent data points –** There must be a minimum of 100 data points from the last 24 hours. The data points don't need to be continuous and can be scattered throughout the 24 hour time frame.
- **Historical data –** There must be a minimum of 100 data points spanning the time frame from 15 days before the current date to 1 day before the current date. The data points don't need to be continuous and can be scattered throughout the 15 day time frame.

> **Note**
>
> For some metrics, Application Insights delays the creation of alarms until the preceding conditions are met. In this case, you get a configuration history event that the metric lacks sufficient data points to establish the alarm threshold.

Feedback

Feedback

You can provide feedback on the automatically generated insights on detected problems by designating them useful or not useful. Your feedback on the insights, along with your application diagnostics (metric anomalies and log exceptions), are used to improve the future detection of similar problems.

Configuration errors

CloudWatch Application Insights uses your configuration to create monitoring telemetries for the components. When Application Insights detects an issue with your account or your configuration, information is provided in the **Remarks** field of the Application summary about how to resolve the configuration issue for your application.

The following table shows suggested resolutions for specific remarks.

Remarks	Suggested resolution	Additional notes
The quota for CloudForm ation has already been reached.	Application Insights creates one CloudFormation stack for each application to manage CloudWatch agent installat ion and configuration for all application component s. By default, each Amazon account can have 2000 stacks. See [Amazon CloudFormation Limits](). To resolve this, raise the limit for CloudFormation stacks.	n/a
No SSM instance role on the following instances.	For Application Insights to be able to install and configure CloudWatch agent on application instances , AmazonSSMManagedIn stanceCore and CloudWatc hAgentServerPolicy policies must be attached to the instance role.	Application Insights calls the SSM [DescribeInstanceIn formation API]() to get the list of instances with SSM permission. After the role is attached to the instance, it takes time for SSM to include the instance in the DescribeInstanceInformation result. Until SSM includes the instance in the result,

Remarks	Suggested resolution	Additional notes
		NO_SSM_INSTANCE_ROLE error remains present for the application.
New components may need configuration.	Application Insights detects that there are new components in the applicati on Resource Group. To resolve this, configure the new components accordingly.	n/a

Logs and metrics supported by Amazon CloudWatch Application Insights

The following lists show the supported logs and metrics for Amazon CloudWatch Application Insights.

CloudWatch Application Insights supports the following logs:

- Microsoft Internet Information Services (IIS) logs
- Error log for SQL Server on EC2
- Custom .NET application logs, such as Log4Net
- Windows Event logs, including Windows logs (System, Application, and Security) and Applications and Services log
- Amazon CloudWatch Logs for Amazon Lambda
- Error log and slow log for RDS MySQL, Aurora MySQL, and MySQL on EC2
- Postgresql log for PostgreSQL RDS and PostgreSQL on EC2
- Amazon CloudWatch Logs for Amazon Step Functions
- Execution logs and access logs (JSON, CSV, and XML, but not CLF) for API Gateway REST API stages
- Prometheus JMX exporter logs (EMF)
- Alert logs and listener logs for Oracle on Amazon RDS and Oracle on Amazon EC2

- Container logs routing from Amazon ECS containers to CloudWatch using `awslogs` log driver.
- Container logs routing from Amazon ECS containers to CloudWatch using FireLens container log router.
- Container logs routing from Amazon EKS or Kubernetes running on Amazon EC2 to CloudWatch using Fluent Bit or Fluentd log processor with Container Insights.
- SAP HANA trace and error logs
- HA Pacemaker logs
- SAP ASE server logs
- SAP ASE backup server logs
- SAP ASE Replication server logs
- SAP ASE RMA agent logs
- SAP ASE Fault Manager logs
- SAP NetWeaver dev trace logs
- Process metrics for Windows processes using proctstat plugin for CloudWatch agent
- Public DNS query logs for hosted zone
- Amazon Route 53 Resolver DNS query logs

CloudWatch Application Insights supports the following log classes:

- **Standard** – Amazon CloudWatch Application Insights requires that log groups are configured with the CloudWatch Logs Standard log class to enable monitoring.

CloudWatch Application Insights supports metrics for the following application components:

- Amazon Elastic Compute Cloud (EC2)
  - CloudWatch built-in metrics
  - CloudWatch agent metrics (Windows server)
  - CloudWatch agent process metrics (Windows server)
  - CloudWatch agent metrics (Linux server)
- Elastic Block Store (EBS)
- Amazon Elastic File System (Amazon EFS)
- Elastic Load Balancer (ELB)

- Application ELB
- Amazon EC2 Auto Scaling groups
- Amazon Simple Queue Server (SQS)
- Amazon Relational Database Service (RDS)
  - RDS Database instances
  - RDS Database clusters
- Amazon Lambda function
- Amazon DynamoDB table
- Amazon S3 bucket
- Amazon Step Functions
  - Execution-level
  - Activity
  - Lambda function
  - Service integration
  - Step Functions API
- API Gateway REST API stages
- SAP HANA
- SAP ASE
- SAP ASE High Availability on Amazon EC2
- SAP NetWeaver
- HA Cluster
- Java
- Amazon Elastic Container Service (Amazon ECS)
  - CloudWatch built-in metrics
  - Container Insights metrics
  - Container Insights Prometheus metrics
- Kubernetes on Amazon
  - Container Insights metrics
  - Container Insights Prometheus metrics
- Amazon FSx

- Amazon VPC
- Amazon VPC NAT gateways
- Amazon Route 53 health check
- Amazon Route 53 hosted zone
- Amazon Route 53 Resolver endpoint
- Amazon Network Firewall rule group
- Amazon Network Firewall rule group association
- Metrics with data points requirements
  - AWS/ApplicationELB
  - AWS/AutoScaling
  - AWS/EC2
  - Elastic Block Store (EBS)
  - AWS/ELB
  - AWS/RDS
  - AWS/Lambda
  - AWS/SQS
  - AWS/CWAgent
  - AWS/DynamoDB
  - AWS/S3
  - AWS/States
  - AWS/ApiGateway
  - AWS/SNS
- Recommended metrics
- Performance Counter metrics

Amazon Elastic Compute Cloud (EC2)

CloudWatch Application Insights supports the following metrics:

Metrics

- CloudWatch built-in metrics

- CloudWatch agent metrics (Windows server)
- CloudWatch agent process metrics (Windows server)
- CloudWatch agent metrics (Linux server)

CloudWatch built-in metrics

CPUCreditBalance

CPUCreditUsage

CPUSurplusCreditBalance

CPUSurplusCreditsCharged

CPUUtilization

DiskReadBytes

DiskReadOps

DiskWriteBytes

DiskWriteOps

EBSByteBalance%

EBSIOBalance%

EBSReadBytes

EBSReadOps

EBSWriteBytes

EBSWriteOps

NetworkIn

NetworkOut

NetworkPacketsIn

NetworkPacketsOut

StatusCheckFailed

StatusCheckFailed_Instance

StatusCheckFailed_System

CloudWatch agent metrics (Windows server)

.NET CLR Exceptions # of Exceps Thrown

.NET CLR Exceptions # of Exceps Thrown/Sec

.NET CLR Exceptions # of Filters/sec

.NET CLR Exceptions # of Finallys/sec

.NET CLR Exceptions Throw to Catch Depth/sec

.NET CLR Interop # of CCWs

.NET CLR Interop # of Stubs

.NET CLR Interop # of TLB exports/sec

.NET CLR Interop # of TLB imports/sec

.NET CLR Interop # of marshaling

.NET CLR Jit % Time in Jit

.NET CLR Jit Standard Jit Failures

.NET CLR Loading % Time Loading

.NET CLR Loading Rate of Load Failures

.NET CLR LocksAndThreads Contention Rate/sec

.NET CLR LocksAndThreads Queue Length/sec

.NET CLR Memory # Total Committed Bytes

.NET CLR Memory % Time in GC

.NET CLR Networking 4.0.0.0 HttpWebRequest Average Queue Time

.NET CLR Networking 4.0.0.0 HttpWebRequests Aborted/sec

.NET CLR Networking 4.0.0.0 HttpWebRequests Failed/sec

.NET CLR Networking 4.0.0.0 HttpWebRequests Queued/sec

APP_POOL_WAS Total Worker Process Ping Failures

ASP.NET Application Restarts

ASP.NET Applications % Managed Processor Time (estimated)

ASP.NET Applications Errors Total/Sec

ASP.NET Applications Errors Unhandled During Execution/sec

ASP.NET Applications Requests in Application Queue

ASP.NET Applications Requests/Sec

ASP.NET Request Wait Time

ASP.NET Requests Queued

HTTP Service Request Queues CurrentQueueSize

LogicalDisk % Free Space

Memory % Committed Bytes In Use

Memory Available Mbytes

Memory Pages/sec

Network Interface Bytes Total/sec

Paging File % Usage

PhysicalDisk % Disk Time

PhysicalDisk Avg. Disk Queue Length

PhysicalDisk Avg. Disk sec/Read

PhysicalDisk Avg. Disk sec/Write

PhysicalDisk Disk Read Bytes/sec

PhysicalDisk Disk Reads/sec

PhysicalDisk Disk Write Bytes/sec

PhysicalDisk Disk Writes/sec

Processor % Idle Time

Processor % Interrupt Time

Processor % Processor Time

Processor % User Time

SQLServer:Access Methods Forwarded Records/sec

SQLServer:Access Methods Full Scans/sec

SQLServer:Access Methods Page Splits/sec

SQLServer:Buffer Manager Buffer cache hit ratio

SQLServer:Buffer Manager Page life expectancy

SQLServer:General Statistics Processes blocked

SQLServer:General Statistics User Connections

SQLServer:Latches Average Latch Wait Time (ms)

SQLServer:Locks Average Wait Time (ms)

SQLServer:Locks Lock Timeouts/sec

SQLServer:Locks Lock Waits/sec

SQLServer:Locks Number of Deadlocks/sec

SQLServer:Memory Manager Memory Grants Pending

SQLServer:SQL Statistics Batch Requests/sec

SQLServer:SQL Statistics SQL Compilations/sec

SQLServer:SQL Statistics SQL Re-Compilations/sec

System Processor Queue Length

TCPv4 Connections Established

TCPv6 Connections Established

W3SVC_W3WP File Cache Flushes

W3SVC_W3WP File Cache Misses

W3SVC_W3WP Requests/Sec

W3SVC_W3WP URI Cache Flushes

W3SVC_W3WP URI Cache Misses

Web Service Bytes Received/Sec

Web Service Bytes Sent/Sec

Web Service Connection attempts/sec

Web Service Current Connections

Web Service Get Requests/sec

Web Service Post Requests/sec

Bytes Received/sec

Normal Messages Queue Length/sec

Urgent Message Queue Length/sec

Reconnect Count

Unacknowledged Message Queue Length/sec

Messages Outstanding

Messages Sent/sec

Database Update Messages/sec

Update Messages/sec

Flushes/sec

Crypto Checkpoints Saved/sec

Crypto Checkpoints Restored/sec

Registry Checkpoints Restored/sec

Registry Checkpoints Saved/sec

Cluster API Calls/sec

Resource API Calls/sec

Cluster Handles/sec

Resource Handles/sec

CloudWatch agent process metrics (Windows server)

Process metrics are collected using the CloudWatch agent procstat plugin. Only Amazon EC2 instances running Windows workloads support process metrics.

procstat cpu_time_system

procstat cpu_time_user

procstat cpu_usage

procstat memory_rss

procstat memory_vms

procstat read_bytes

procstat write_bytes

.procstat read_count

procstat write_count

CloudWatch agent metrics (Linux server)

cpu_time_active

cpu_time_guest

cpu_time_guest_nice

cpu_time_idle

cpu_time_iowait

cpu_time_irq

cpu_time_nice

cpu_time_softirq

cpu_time_steal

cpu_time_system

cpu_time_user

cpu_usage_active

cpu_usage_guest

cpu_usage_guest_nice

cpu_usage_idle

cpu_usage_iowait

cpu_usage_irq

cpu_usage_nice

cpu_usage_softirq

cpu_usage_steal

cpu_usage_system

cpu_usage_user

disk_free

disk_inodes_free

disk_inodes_used

disk_used

disk_used_percent

diskio_io_time

diskio_iops_in_progress

diskio_read_bytes

diskio_read_time

diskio_reads

diskio_write_bytes

diskio_write_time

diskio_writes

mem_active

mem_available

mem_available_percent

mem_buffered

mem_cached

mem_free

mem_inactive

mem_used

mem_used_percent

net_bytes_recv

net_bytes_sent

net_drop_in

net_drop_out

net_err_in

net_err_out

net_packets_recv

net_packets_sent

netstat_tcp_close

netstat_tcp_close_wait

netstat_tcp_closing

netstat_tcp_established

netstat_tcp_fin_wait1

netstat_tcp_fin_wait2

netstat_tcp_last_ack

netstat_tcp_listen

netstat_tcp_none

netstat_tcp_syn_recv

netstat_tcp_syn_sent

netstat_tcp_time_wait

netstat_udp_socket

processes_blocked

processes_dead

processes_idle

processes_paging

processes_running

processes_sleeping

processes_stopped

processes_total

processes_total_threads

processes_wait

processes_zombies

swap_free

swap_used

swap_used_percent

Elastic Block Store (EBS)

CloudWatch Application Insights supports the following metrics:

VolumeReadBytes

VolumeWriteBytes

VolumeReadOps

VolumeWriteOps

VolumeTotalReadTime

VolumeTotalWriteTime

VolumeIdleTime

VolumeQueueLength

VolumeThroughputPercentage

VolumeConsumedReadWriteOps

BurstBalance

Amazon Elastic File System (Amazon EFS)

CloudWatch Application Insights supports the following metrics:

BurstCreditBalance

PercentIOLimit

PermittedThroughput

MeteredIOBytes

TotalIOBytes

DataWriteIOBytes

DataReadIOBytes

MetadataIOBytes

ClientConnections

TimeSinceLastSync

StorageBytes

Throughput

PercentageOfPermittedThroughputUtilization

ThroughputIOPS

PercentThroughputDataReadIOByte

PercentThroughputDataWriteIOBytes

PercentageOfIOPSDataReadIOBytes

PercentageOfIOPSDataWriteIOBytes

AverageDataReadIOBytesSize

AverageDataWriteIOBytesSize

Elastic Load Balancer (ELB)

CloudWatch Application Insights supports the following metrics:

EstimatedALBActiveConnectionCount

EstimatedALBConsumedLCUs

EstimatedALBNewConnectionCount

EstimatedProcessedBytes

HTTPCode_Backend_4XX

HTTPCode_Backend_5XX

HealthyHostCount

RequestCount

UnHealthyHostCount

Application ELB

CloudWatch Application Insights supports the following metrics:

EstimatedALBActiveConnectionCount

EstimatedALBConsumedLCUs

EstimatedALBNewConnectionCount

EstimatedProcessedBytes

HTTPCode_Backend_4XX

HTTPCode_Backend_5XX

HealthyHostCount

Latency

RequestCount

SurgeQueueLength

UnHealthyHostCount

Amazon EC2 Auto Scaling groups

CloudWatch Application Insights supports the following metrics:

CPUCreditBalance

CPUCreditUsage

CPUSurplusCreditBalance

CPUSurplusCreditsCharged

CPUUtilization

DiskReadBytes

DiskReadOps

DiskWriteBytes

DiskWriteOps

EBSByteBalance%

EBSIOBalance%

EBSReadBytes

EBSReadOps

EBSWriteBytes

EBSWriteOps

NetworkIn

NetworkOut

NetworkPacketsIn

NetworkPacketsOut

StatusCheckFailed

StatusCheckFailed_Instance

StatusCheckFailed_System

Amazon Simple Queue Server (SQS)

CloudWatch Application Insights supports the following metrics:

ApproximateAgeOfOldestMessage

ApproximateNumberOfMessagesDelayed

ApproximateNumberOfMessagesNotVisible

ApproximateNumberOfMessagesVisible

NumberOfEmptyReceives

NumberOfMessagesDeleted

NumberOfMessagesReceived

NumberOfMessagesSent

Amazon Relational Database Service (RDS)

CloudWatch Application Insights supports the following metrics:

Metrics

- RDS Database instances
- RDS Database clusters

RDS Database instances

BurstBalance

CPUCreditBalance

CPUUtilization

DatabaseConnections

DiskQueueDepth

FailedSQLServerAgentJobsCount

FreeStorageSpace

FreeableMemory

NetworkReceiveThroughput

NetworkTransmitThroughput

ReadIOPS

ReadLatency

ReadThroughput

WriteIOPS

WriteLatency

WriteThroughput

RDS Database clusters

ActiveTransactions

AuroraBinlogReplicaLag

AuroraReplicaLag

BackupRetentionPeriodStorageUsed

BinLogDiskUsage

BlockedTransactions

BufferCacheHitRatio

CPUUtilization

CommitLatency

CommitThroughput

DDLLatency

DDLThroughput

DMLLatency

DMLThroughput

DatabaseConnections

Deadlocks

DeleteLatency

DeleteThroughput

EngineUptime

FreeLocalStorage

FreeableMemory

InsertLatency

InsertThroughput

LoginFailures

NetworkReceiveThroughput

NetworkThroughput

NetworkTransmitThroughput

Queries

ResultSetCacheHitRatio

SelectLatency

SelectThroughput

SnapshotStorageUsed

TotalBackupStorageBilled

UpdateLatency

UpdateThroughput

VolumeBytesUsed

VolumeReadIOPs

VolumeWriteIOPs

Amazon Lambda function

CloudWatch Application Insights supports the following metrics:

Errors

DeadLetterErrors

Duration

Throttles

IteratorAge

ProvisionedConcurrencySpilloverInvocations

Amazon DynamoDB table

CloudWatch Application Insights supports the following metrics:

SystemErrors

UserErrors

ConsumedReadCapacityUnits

ConsumedWriteCapacityUnits

ReadThrottleEvents

WriteThrottleEvents

TimeToLiveDeletedItemCount

ConditionalCheckFailedRequests

TransactionConflict

ReturnedRecordsCount

PendingReplicationCount

ReplicationLatency

Amazon S3 bucket

CloudWatch Application Insights supports the following metrics:

ReplicationLatency

BytesPendingReplication

OperationsPendingReplication

4xxErrors

5xxErrors

AllRequests

GetRequests

PutRequests

DeleteRequests

HeadRequests

PostRequests

SelectRequests

ListRequests

SelectScannedBytes

SelectReturnedBytes

FirstByteLatency

TotalRequestLatency

BytesDownloaded

BytesUploaded

Amazon Step Functions

CloudWatch Application Insights supports the following metrics:

Metrics

- Execution-level
- Activity
- Lambda function
- Service integration
- Step Functions API

Execution-level

ExecutionTime

ExecutionThrottled

ExecutionsFailed

ExecutionsTimedOut

ExecutionsAborted

ExecutionsSucceeded

ExecutionsStarted

Activity

ActivityRunTime

ActivityScheduleTime

ActivityTime

ActivitiesFailed

ActivitiesHeartbeatTimedOut

ActivitiesTimedOut

ActivitiesScheduled

ActivitiesSucceeded

ActivitiesStarted

Lambda function

LambdaFunctionRunTime

LambdaFunctionScheduleTime

LambdaFunctionTime

LambdaFunctionsFailed

LambdaFunctionsTimedOut

LambdaFunctionsScheduled

LambdaFunctionsSucceeded

LambdaFunctionsStarted

Service integration

ServiceIntegrationRunTime

ServiceIntegrationScheduleTime

ServiceIntegrationTime

ServiceIntegrationsFailed

ServiceIntegrationsTimedOut

ServiceIntegrationsScheduled

ServiceIntegrationsSucceeded

ServiceIntegrationsStarted

Step Functions API

ThrottledEvents

ProvisionedBucketSize

ProvisionedRefillRate

ConsumedCapacity

API Gateway REST API stages

CloudWatch Application Insights supports the following metrics:

4XXError

5XXError

IntegrationLatency

Latency

CacheHitCount

CacheMissCount

SAP HANA

> **Note**
>
> CloudWatch Application Insights supports only single SID HANA environments. If multiple HANA SIDs are attached, monitoring will be set up for only the first detected SID.

CloudWatch Application Insights supports the following metrics:

hanadb_every_service_started_status

hanadb_daemon_service_started_status

hanadb_preprocessor_service_started_status

hanadb_webdispatcher_service_started_status

hanadb_compileserver_service_started_status

hanadb_nameserver_service_started_status

hanadb_server_startup_time_variations_seconds

hanadb_level_5_alerts_count

hanadb_level_4_alerts_count

hanadb_out_of_memory_events_count

hanadb_max_trigger_read_ratio_percent

hanadb_max_trigger_write_ratio_percent

hanadb_log_switch_wait_ratio_percent

hanadb_log_switch_race_ratio_percent

hanadb_time_since_last_savepoint_seconds

hanadb_disk_usage_highlevel_percent

hanadb_max_converter_page_number_count

hanadb_long_running_savepoints_count

hanadb_failed_io_reads_count

hanadb_failed_io_writes_count

hanadb_disk_data_unused_percent

hanadb_current_allocation_limit_used_percent

hanadb_table_allocation_limit_used_percent

hanadb_host_total_physical_memory_mb

hanadb_host_physical_memory_used_mb

hanadb_host_physical_memory_free_mb

hanadb_swap_memory_free_mb

hanadb_swap_memory_used_mb

hanadb_host_allocation_limit_mb

hanadb_host_total_memory_used_mb

hanadb_host_total_peak_memory_used_mb

hanadb_host_total_allocation_limit_mb

hanadb_host_code_size_mb

hanadb_host_shared_memory_allocation_mb

hanadb_cpu_usage_percent

hanadb_cpu_user_percent

hanadb_cpu_system_percent

hanadb_cpu_waitio_percent

hanadb_cpu_busy_percent

hanadb_cpu_idle_percent

hanadb_long_delta_merge_count

hanadb_unsuccessful_delta_merge_count

hanadb_successful_delta_merge_count

hanadb_row_store_allocated_size_mb

hanadb_row_store_free_size_mb

hanadb_row_store_used_size_mb

hanadb_temporary_tables_count

hanadb_large_non_compressed_tables_count

hanadb_total_non_compressed_tables_count

hanadb_longest_running_job_seconds

hanadb_average_commit_time_milliseconds

hanadb_suspended_sql_statements_count

hanadb_plan_cache_hit_ratio_percent

hanadb_plan_cache_lookup_count

hanadb_plan_cache_hit_count

hanadb_plan_cache_total_execution_microseconds

hanadb_plan_cache_cursor_duration_microseconds

hanadb_plan_cache_preparation_microseconds

hanadb_plan_cache_evicted_count

hanadb_plan_cache_evicted_microseconds

hanadb_plan_cache_evicted_preparation_count

hanadb_plan_cache_evicted_execution_count

hanadb_plan_cache_evicted_preparation_microseconds

hanadb_plan_cache_evicted_cursor_duration_microseconds hanadb_plan_cache_evicted_total_execution_microseconds

hanadb_plan_cache_evicted_plan_size_mb

hanadb_plan_cache_count

hanadb_plan_cache_preparation_count

hanadb_plan_cache_execution_count

hanadb_network_collision_rate

hanadb_network_receive_rate

hanadb_network_transmit_rate

hanadb_network_packet_receive_rate

hanadb_network_packet_transmit_rate

hanadb_network_transmit_error_rate

hanadb_network_receive_error_rate

hanadb_time_until_license_expires_days

hanadb_is_license_valid_status

hanadb_local_running_connections_count

hanadb_local_idle_connections_count

hanadb_remote_running_connections_count

hanadb_remote_idle_connections_count

hanadb_last_full_data_backup_age_days

hanadb_last_data_backup_age_days

hanadb_last_log_backup_age_hours

hanadb_failed_data_backup_past_7_days_count hanadb_failed_log_backup_past_7_days_count

hanadb_oldest_backup_in_catalog_age_days

hanadb_backup_catalog_size_mb

hanadb_hsr_replication_status

hanadb_hsr_log_shipping_delay_seconds

hanadb_hsr_secondary_failover_count

hanadb_hsr_secondary_reconnect_count

hanadb_hsr_async_buffer_used_mb

hanadb_hsr_secondary_active_status

hanadb_handle_count

hanadb_ping_time_milliseconds

hanadb_connection_count

hanadb_internal_connection_count

hanadb_external_connection_count

hanadb_idle_connection_count

hanadb_transaction_count

hanadb_internal_transaction_count

hanadb_external_transaction_count

hanadb_user_transaction_count

hanadb_blocked_transaction_count

hanadb_statement_count

hanadb_active_commit_id_range_count

hanadb_mvcc_version_count

hanadb_pending_session_count

hanadb_record_lock_count

hanadb_read_count

hanadb_write_count

hanadb_merge_count

hanadb_unload_count

hanadb_active_thread_count

hanadb_waiting_thread_count

hanadb_total_thread_count

hanadb_active_sql_executor_count

hanadb_waiting_sql_executor_count

hanadb_total_sql_executor_count

hanadb_data_write_size_mb

hanadb_data_write_time_milliseconds

hanadb_log_write_size_mb

hanadb_log_write_time_milliseconds

hanadb_data_read_size_mb

hanadb_data_read_time_milliseconds

hanadb_log_read_size_mb

hanadb_log_read_time_milliseconds

hanadb_data_backup_write_size_mb

hanadb_data_backup_write_time_milliseconds

hanadb_log_backup_write_size_mb

hanadb_log_backup_write_time_milliseconds

hanadb_mutex_collision_count

hanadb_read_write_lock_collision_count

hanadb_admission_control_admit_count

hanadb_admission_control_reject_count

hanadb_admission_control_queue_size_mb

hanadb_admission_control_wait_time_milliseconds

SAP ASE

CloudWatch Application Insights supports the following metrics:

asedb_database_availability

asedb_trunc_log_on_chkpt_enabled

asedb_last_db_backup_age_in_days

asedb_last_transaction_log_backup_age_in_hours

asedb_suspected_database

asedb_db_space_usage_percent

asedb_db_log_space_usage_percent

asedb_locked_login

asedb_has_mixed_log_and_data

asedb_runtime_for_open_transactions

asedb_data_cache_hit_ratio

asedb_data_cache_usage

asedb_sql_cache_hit_ratio

asedb_cache_usage

asedb_run_queue_length

asedb_number_of_rollbacks

asedb_number_of_commits

asedb_number_of_transactions

asedb_outstanding_disk_io

asedb_percent_io_busy

asedb_percent_system_busy

asedb_percent_locks_active

asedb_scheduled_jobs_failed_percent

asedb_user_connections_percent

asedb_query_logical_reads

asedb_query_physical_reads

asedb_query_cpu_time

asedb_query_memory_usage

SAP ASE High Availability on Amazon EC2

CloudWatch Application Insights supports the following metrics:

asedb_ha_replication_state

asedb_ha_replication_mode

asedb_ha_replication_latency_in_minutes

SAP NetWeaver

CloudWatch Application Insights supports the following metrics:

Metric	Description
sap_alerts_ResponseTime	The SAP response time alert from CCMS (RZ20)>R3Services>Dialog>ResponseTime.
sap_alerts_ResponseTimeDialog	The SAP response time dialog alert from CCMS (RZ20)>R3Services>Dialog> ResponseTimeDialog.
sap_alerts_ResponseTimeDialogRFC	The SAP response time alert from CCMS (RZ20)>R3Services> Dialog>ResponseTimeDialogRFC.
sap_alerts_DBRequestTime	The SAP response time alert from CCMS (RZ20)>R3Services>Dialog>DBRequestTime.
sap_alerts_FrontendResponseTime	The SAP response time alert from CCMS (RZ20)>R3Services > Dialog>FrontEndResponseTime.
sap_alerts_Database	The SAP system has logged database-related errors. Alert from SM21 or CCMS (RZ20)>R3 Syslog>Database.
sap_alerts_QueueTime	The SAP queue time alert from CCMS (RZ20)>R3Services>Dialog>QueueTime.
sap_alerts_AbortedJobs	Failed background jobs in SAP system. Alert from (RZ20)>R3Services > Background>AbortedJobs.

Metric	Description
sap_alerts_BasisSystem	SAP system logged system-level errors. Alert from SM21 or CCMS (RZ20)>R3Syslog>Ba sisSystem.
sap_alerts_Security	The SAP system logged security-related messages. Alert from SM21 or CCMS (RZ20)>R3Syslog>Security.
sap_alerts_System	The SAP system logged security or audit-rel ated messages. Alert from SM21 or CCMS (RZ20)>Security>System.
sap_alerts_LongRunners	There are long running programs in your SAP system. Alert from CCMS (RZ20)>R3Services > Dialog>LongRunners.
sap_alerts_SqlError	There are SAP database client layer error logs. Alert from CCMS(RZ20)>DatabaseClient>A bapSql>SqlError.
sap_alerts_State	State alert from CCMS (RZ20)>OS Collector >State.
sap_alerts_Shortdumps	Shortdumps alert from ST22 and CCMS (RZ20)>R3Abap>Shortdumps.
sap_alerts_Availability	Availability alert for SAP application server instance from SM21, SM50, SM51, SM66, and CCMS (RZ20)>InstanceAsTask>Availability.
sap_dispatcher_queue_high	The SAPControl Web Service function `GetQueueStatistic` provides the dispatcher queue high count.
sap_dispatcher_queue_max	The SAPControl Web Service function `GetQueueStatistic` provides the dispatcher queue max count.

Metric	Description
sap_dispatcher_queue_now	The SAPControl Web Service function `GetQueueStatistic` provides the dispatcher queue now count.
sap_dispatcher_queue_reads	The SAPControl Web Service function `GetQueueStatistic` provides the dispatcher queue reads count.
sap_dispatcher_queue_writes	The SAPControl Web Service function `GetQueueStatistic` provides the dispatcher queue writes count.
sap_enqueue_server_arguments_high	The SAPControl Web Service function `EnqGetStatistic` provides the enqueue arguments high.
sap_enqueue_server_arguments_max	The SAPControl Web Service function `EnqGetStatistic` provides the enqueue arguments max.
sap_enqueue_server_arguments_now	The SAPControl Web Service function `EnqGetStatistic` provides the enqueue arguments now.
sap_enqueue_server_arguments_state	The SAPControl Web Service function `EnqGetStatistic` provides the enqueue arguments state.
sap_enqueue_server_backup_requests	The SAPControl Web Service function `EnqGetStatistic` provides the enqueue backup requests.
sap_enqueue_server_cleanup_requests	The SAPControl Web Service function `EnqGetStatistic` provides the enqueue cleanup requests.

Metric	Description
sap_enqueue_server_dequeue_all_requests	The SAPControl Web Service function `EnqGetStatistic` provides the dequeue all requests.
sap_enqueue_server_dequeue_errors	The SAPControl Web Service function `EnqGetStatistic` provides the dequeue errors.
sap_enqueue_server_dequeue_requests	The SAPControl Web Service function `EnqGetStatistic` provides the dequeue requests.
sap_enqueue_server_enqueue_errors	The SAPControl Web Service function `EnqGetStatistic` provides the enqueue errors.
sap_enqueue_server_enqueue_rejects	The SAPControl Web Service function `EnqGetStatistic` provides the enqueue rejects.
sap_enqueue_server_enqueue_requests	The SAPControl Web Service function `EnqGetStatistic` provides the enqueue requests.
sap_enqueue_server_lock_time	The SAPControl Web Service function `EnqGetStatistic` provides the enqueue lock time.
sap_enqueue_server_lock_wait_time	The SAPControl Web Service function `EnqGetStatistic` provides the enqueue lock wait time.
sap_enqueue_server_locks_high	The SAPControl Web Service function `EnqGetStatistic` provides the enqueue locks high.

Metric	Description
sap_enqueue_server_locks_max	The SAPControl Web Service function `EnqGetStatistic` provides the enqueue locks max.
sap_enqueue_server_locks_now	The SAPControl Web Service function `EnqGetStatistic` provides the enqueue locks now.
sap_enqueue_server_locks_state	The SAPControl Web Service function `EnqGetStatistic` provides the enqueue locks state.
sap_enqueue_server_owner_high	The SAPControl Web Service function `EnqGetStatistic` provides the enqueue owner high.
sap_enqueue_server_owner_max	The SAPControl Web Service function `EnqGetStatistic` provides the enqueue owner max.
sap_enqueue_server_owner_now	The SAPControl Web Service function `EnqGetStatistic` provides the enqueue owner now.
sap_enqueue_server_owner_state	The SAPControl Web Service function `EnqGetStatistic` provides the enqueue owner state.
sap_enqueue_server_replication_state	The SAPControl Web Service function `EnqGetStatistic` provides the enqueue replication state status.
sap_enqueue_server_reporting_requests	The SAPControl Web Service function `EnqGetStatistic` provides the reqporting requests status.

Metric	Description
sap_enqueue_server_server_time	The SAPControl Web Service function `EnqGetStatistic` provides the enqueue server time.
sap_HA_check_failover_config_state	The SAPControl Web Service function `HACheckFailoverConfig` provides the SAP High Availability status.
sap_HA_get_failover_config_HAActive	The SAPControl Web Service function `HAGetFailoverConfig` provides the SAP High Availability Cluster configuration and status.
sap_start_service_processes	The SAPControl Web Service function `GetProcessList` provides the disp+work , IGS, gwrd, icman, message server, and enqueue server processes status.

HA Cluster

CloudWatch Application Insights supports the following metrics:

ha_cluster_pacemaker_stonith_enabled

ha_cluster_corosync_quorate

hanadb_webdispatcher_service_started_status

ha_cluster_pacemaker_nodes

ha_cluster_corosync_ring_errors

ha_cluster_pacemaker_fail_count

Java

CloudWatch Application Insights supports the following metrics:

java_lang_memory_heapmemoryusage_used

java_lang_memory_heapmemoryusage_committed

java_lang_operatingsystem_openfiledescriptorcount

java_lang_operatingsystem_maxfiledescriptorcount

java_lang_operatingsystem_freephysicalmemorysize

java_lang_operatingsystem_freeswapspacesize

java_lang_threading_threadcount

java_lang_threading_daemonthreadcount

java_lang_classloading_loadedclasscount

java_lang_garbagecollector_collectiontime_copy

java_lang_garbagecollector_collectiontime_ps_scavenge

java_lang_garbagecollector_collectiontime_parnew

java_lang_garbagecollector_collectiontime_marksweepcompact

java_lang_garbagecollector_collectiontime_ps_marksweep

java_lang_garbagecollector_collectiontime_concurrentmarksweep

java_lang_garbagecollector_collectiontime_g1_young_generation

java_lang_garbagecollector_collectiontime_g1_old_generation

java_lang_garbagecollector_collectiontime_g1_mixed_generation

java_lang_operatingsystem_committedvirtualmemorysize

Amazon Elastic Container Service (Amazon ECS)

CloudWatch Application Insights supports the following metrics:

Metrics

- CloudWatch built-in metrics

- Container Insights metrics
- Container Insights Prometheus metrics

CloudWatch built-in metrics

CPUReservation

CPUUtilization

MemoryReservation

MemoryUtilization

GPUReservation

Container Insights metrics

ContainerInstanceCount

CpuUtilized

CpuReserved

DeploymentCount

DesiredTaskCount

MemoryUtilized

MemoryReserved

NetworkRxBytes

NetworkTxBytes

PendingTaskCount

RunningTaskCount

ServiceCount

StorageReadBytes

StorageWriteBytes

TaskCount

TaskSetCount

instance_cpu_limit

instance_cpu_reserved_capacity

instance_cpu_usage_total

instance_cpu_utilization

instance_filesystem_utilization

instance_memory_limit

instance_memory_reserved_capacity

instance_memory_utilization

instance_memory_working_set

instance_network_total_bytes

instance_number_of_running_tasks

Container Insights Prometheus metrics

Java JMX metrics

java_lang_memory_heapmemoryusage_used

java_lang_memory_heapmemoryusage_committed

java_lang_operatingsystem_openfiledescriptorcount

java_lang_operatingsystem_maxfiledescriptorcount

java_lang_operatingsystem_freephysicalmemorysize

java_lang_operatingsystem_freeswapspacesize java_lang_threading_threadcount

java_lang_classloading_loadedclasscount

java_lang_threading_daemonthreadcount

java_lang_garbagecollector_collectiontime_copy

java_lang_garbagecollector_collectiontime_ps_scavenge

java_lang_garbagecollector_collectiontime_parnew

java_lang_garbagecollector_collectiontime_marksweepcompact

java_lang_garbagecollector_collectiontime_ps_marksweep

java_lang_garbagecollector_collectiontime_concurrentmarksweep

java_lang_garbagecollector_collectiontime_g1_young_generation

java_lang_garbagecollector_collectiontime_g1_old_generation

java_lang_garbagecollector_collectiontime_g1_mixed_generation

java_lang_operatingsystem_committedvirtualmemorysize

Kubernetes on Amazon

CloudWatch Application Insights supports the following metrics:

Metrics

- Container Insights metrics
- Container Insights Prometheus metrics

Container Insights metrics

cluster_failed_node_count

cluster_node_count

namespace_number_of_running_pods

node_cpu_limit

node_cpu_reserved_capacity

node_cpu_usage_total

node_cpu_utilization

node_filesystem_utilization

node_memory_limit

node_memory_reserved_capacity

node_memory_utilization

node_memory_working_set

node_network_total_bytes

node_number_of_running_containers

node_number_of_running_pods

pod_cpu_reserved_capacity

pod_cpu_utilization

pod_cpu_utilization_over_pod_limit

pod_memory_reserved_capacity

pod_memory_utilization

pod_memory_utilization_over_pod_limit

pod_network_rx_bytes

pod_network_tx_bytes

service_number_of_running_pods

Container Insights Prometheus metrics

Java JMX metrics

java_lang_memory_heapmemoryusage_used

java_lang_memory_heapmemoryusage_committed

java_lang_operatingsystem_openfiledescriptorcount

java_lang_operatingsystem_maxfiledescriptorcount

java_lang_operatingsystem_freephysicalmemorysize

java_lang_operatingsystem_freeswapspacesize

java_lang_threading_threadcount

java_lang_classloading_loadedclasscount

java_lang_threading_daemonthreadcount

java_lang_garbagecollector_collectiontime_copy

java_lang_garbagecollector_collectiontime_ps_scavenge

java_lang_garbagecollector_collectiontime_parnew

java_lang_garbagecollector_collectiontime_marksweepcompact

java_lang_garbagecollector_collectiontime_ps_marksweep

java_lang_garbagecollector_collectiontime_concurrentmarksweep

java_lang_garbagecollector_collectiontime_g1_young_generation

java_lang_garbagecollector_collectiontime_g1_old_generation

java_lang_garbagecollector_collectiontime_g1_mixed_generation

java_lang_operatingsystem_committedvirtualmemorysize

Amazon FSx

CloudWatch Application Insights supports the following metrics:

DataReadBytes

DataWriteBytes

DataReadOperations

DataWriteOperations

MetadataOperations

FreeStorageCapacity

FreeDataStorageCapacity

LogicalDiskUsage

PhysicalDiskUsage

Amazon VPC

CloudWatch Application Insights supports the following metrics:

NetworkAddressUsage

NetworkAddressUsagePeered

VPCFirewallQueryVolume

Amazon VPC NAT gateways

CloudWatch Application Insights supports the following metrics:

ErrorPortAllocation

IdleTimeoutCount

Amazon Route 53 health check

CloudWatch Application Insights supports the following metrics:

ChildHealthCheckHealthyCount

ConnectionTime

HealthCheckPercentageHealthy

HealthCheckStatus

SSLHandshakeTime

TimeToFirstByte

Amazon Route 53 hosted zone

CloudWatch Application Insights supports the following metrics:

DNSQueries

DNSSECInternalFailure

DNSSECKeySigningKeysNeedingAction

DNSSECKeySigningKeyMaxNeedingActionAge

DNSSECKeySigningKeyAge

Amazon Route 53 Resolver endpoint

CloudWatch Application Insights supports the following metrics:

EndpointHealthyENICount

EndpointUnHealthyENICount

InboundQueryVolume

OutboundQueryVolume

OutboundQueryAggregateVolume

Amazon Network Firewall rule group

CloudWatch Application Insights supports the following metrics:

FirewallRuleGroupQueryVolume

Amazon Network Firewall rule group association

CloudWatch Application Insights supports the following metrics:

FirewallRuleGroupVpcQueryVolume

Metrics with data points requirements

For metrics without an obvious default threshold to alarm on, Application Insights waits until the metric has enough data points to predict a reasonable threshold to alarm on. The metric data points requirement that CloudWatch Application Insights checks before an alarm is created are:

- The metric has at least 100 data points from the past 15 to the past 2 days.
- The metric has at least 100 data points from the last day.

The following metrics follow these data points requirements. Note that CloudWatch agent metrics require up to one hour to create alarms.

Metrics

- AWS/ApplicationELB
- AWS/AutoScaling
- AWS/EC2
- Elastic Block Store (EBS)
- AWS/ELB
- AWS/RDS
- AWS/Lambda
- AWS/SQS
- AWS/CWAgent
- AWS/DynamoDB
- AWS/S3
- AWS/States
- AWS/ApiGateway
- AWS/SNS

AWS/ApplicationELB

ActiveConnectionCount

ConsumedLCUs

HTTPCode_ELB_4XX_Count

HTTPCode_Target_2XX_Count

HTTPCode_Target_3XX_Count

HTTPCode_Target_4XX_Count

HTTPCode_Target_5XX_Count

NewConnectionCount

ProcessedBytes

TargetResponseTime

UnHealthyHostCount

AWS/AutoScaling

GroupDesiredCapacity

GroupInServiceInstances

GroupMaxSize

GroupMinSize

GroupPendingInstances

GroupStandbyInstances

GroupTerminatingInstances

GroupTotalInstances

AWS/EC2

CPUCreditBalance

CPUCreditUsage

CPUSurplusCreditBalance

CPUSurplusCreditsCharged

CPUUtilization

DiskReadBytes

DiskReadOps

DiskWriteBytes

DiskWriteOps

EBSByteBalance%

EBSIOBalance%

EBSReadBytes

EBSReadOps

EBSWriteBytes

EBSWriteOps

NetworkIn

NetworkOut

NetworkPacketsIn

NetworkPacketsOut

Elastic Block Store (EBS)

VolumeReadBytes

VolumeWriteBytes

VolumeReadOps

VolumeWriteOps

VolumeTotalReadTime

VolumeTotalWriteTime

VolumeIdleTime

VolumeQueueLength

VolumeThroughputPercentage

VolumeConsumedReadWriteOps

BurstBalance

AWS/ELB

EstimatedALBActiveConnectionCount

EstimatedALBConsumedLCUs

EstimatedALBNewConnectionCount

EstimatedProcessedBytes

HTTPCode_Backend_4XX

HTTPCode_Backend_5XX

HealthyHostCount

Latency

RequestCount

SurgeQueueLength

UnHealthyHostCount

AWS/RDS

ActiveTransactions

AuroraBinlogReplicaLag

AuroraReplicaLag

BackupRetentionPeriodStorageUsed

BinLogDiskUsage

BlockedTransactions

CPUCreditBalance

CommitLatency

CommitThroughput

DDLLatency

DDLThroughput

DMLLatency

DMLThroughput

DatabaseConnections

Deadlocks

DeleteLatency

DeleteThroughput

DiskQueueDepth

EngineUptime

FreeLocalStorage

FreeStorageSpace

FreeableMemory

InsertLatency

InsertThroughput

LoginFailures

NetworkReceiveThroughput

NetworkThroughput

NetworkTransmitThroughput

Queries

ReadIOPS

ReadThroughput

SelectLatency

SelectThroughput

SnapshotStorageUsed

TotalBackupStorageBilled

UpdateLatency

UpdateThroughput

VolumeBytesUsed

VolumeReadIOPs

VolumeWriteIOPs

WriteIOPS

WriteThroughput

AWS/Lambda

Errors

DeadLetterErrors

Duration

Throttles

IteratorAge

ProvisionedConcurrencySpilloverInvocations

AWS/SQS

ApproximateAgeOfOldestMessage

ApproximateNumberOfMessagesDelayed

ApproximateNumberOfMessagesNotVisible

ApproximateNumberOfMessagesVisible

NumberOfEmptyReceives

NumberOfMessagesDeleted

NumberOfMessagesReceived

NumberOfMessagesSent

AWS/CWAgent

LogicalDisk % Free Space

Memory % Committed Bytes In Use

Memory Available Mbytes

Network Interface Bytes Total/sec

Paging File % Usage

PhysicalDisk % Disk Time

PhysicalDisk Avg. Disk sec/Read

PhysicalDisk Avg. Disk sec/Write

PhysicalDisk Disk Read Bytes/sec

PhysicalDisk Disk Reads/sec

PhysicalDisk Disk Write Bytes/sec

PhysicalDisk Disk Writes/sec

Processor % Idle Time

Processor % Interrupt Time

Processor % Processor Time

Processor % User Time

SQLServer:Access Methods Forwarded Records/sec

SQLServer:Access Methods Page Splits/sec

SQLServer:Buffer Manager Buffer cache hit ratio

SQLServer:Buffer Manager Page life expectancy

SQLServer:Database Replica File Bytes Received/sec

SQLServer:Database Replica Log Bytes Received/sec

SQLServer:Database Replica Log remaining for undo

SQLServer:Database Replica Log Send Queue

SQLServer:Database Replica Mirrored Write Transaction/sec

SQLServer:Database Replica Recovery Queue

SQLServer:Database Replica Redo Bytes Remaining

SQLServer:Database Replica Redone Bytes/sec

SQLServer:Database Replica Total Log requiring undo

SQLServer:Database Replica Transaction Delay

SQLServer:General Statistics Processes blocked

SQLServer:SQL Statistics Batch Requests/sec

SQLServer:SQL Statistics SQL Compilations/sec

SQLServer:SQL Statistics SQL Re-Compilations/sec

System Processor Queue Length

TCPv4 Connections Established

TCPv6 Connections Established

AWS/DynamoDB

ConsumedReadCapacityUnits

ConsumedWriteCapacityUnits

ReadThrottleEvents

WriteThrottleEvents

TimeToLiveDeletedItemCount

ConditionalCheckFailedRequests

TransactionConflict

ReturnedRecordsCount

PendingReplicationCount

ReplicationLatency

AWS/S3

ReplicationLatency

BytesPendingReplication

OperationsPendingReplication

4xxErrors

5xxErrors

AllRequests

GetRequests

PutRequests

DeleteRequests

HeadRequests

PostRequests

SelectRequests

ListRequests

SelectScannedBytes

SelectReturnedBytes

FirstByteLatency

TotalRequestLatency

BytesDownloaded

BytesUploaded

AWS/States

ActivitiesScheduled

ActivitiesStarted

ActivitiesSucceeded

ActivityScheduleTime

ActivityRuntime

ActivityTime

LambdaFunctionsScheduled

LambdaFunctionsStarted

LambdaFunctionsSucceeded

LambdaFunctionScheduleTime

LambdaFunctionRuntime

LambdaFunctionTime

ServiceIntegrationsScheduled

ServiceIntegrationsStarted

ServiceIntegrationsSucceeded

ServiceIntegrationScheduleTime

ServiceIntegrationRuntime

ServiceIntegrationTime

ProvisionedRefillRate

ProvisionedBucketSize

ConsumedCapacity

ThrottledEvents

AWS/ApiGateway

4XXError

IntegrationLatency

Latency

DataProcessed

CacheHitCount

CacheMissCount

AWS/SNS

NumberOfNotificationsDelivered

NumberOfMessagesPublished

NumberOfNotificationsFailed

NumberOfNotificationsFilteredOut

NumberOfNotificationsFilteredOut-InvalidAttributes

NumberOfNotificationsFilteredOut-NoMessageAttributes

NumberOfNotificationsRedrivenToDlq

NumberOfNotificationsFailedToRedriveToDlq

SMSSuccessRate

Recommended metrics

The following table lists the recommended metrics for each component type.

Component type	Workload type	Recommended metric
EC2 instance (Windows servers)	Default/Custom	CPUUtilization StatusCheckFailed Processor % Processor Time Memory % Committed Bytes In Use LogicalDisk % Free Space Memory Available Mbytes
	Active Directory	CPUUtilization StatusCheckFailed Processor % Processor Time Memory % Committed Bytes In Use Memory Available Mbytes Database ==> Instances Database Cache % Hit DirectoryServices DRA Pending Replication Operations DirectoryServices DRA Pending Replication Synchronizations

Component type	Workload type	Recommended metric
		DNS Recursive Query Failure/ sec LogicalDisk Avg. Disk Queue Length
	Java Application	CPUUtilization StatusCheckFailed Processor % Processor Time Memory % Committed Bytes In Use Memory Available Mbytes java_lang_threading_threadc ount java_lang_classloading_load edclasscount java_lang_memory_h eapmemoryusage_used java_lang_memory_h eapmemoryusage_committed java_lang_operatingsystem_f reephysicalmemorysize java_lang_operatingsystem_f reeswapspacesize

Component type	Workload type	Recommended metric
	Microsoft IIS/.NET Web Front-End	CPUUtilization StatusCheckFailed Processor % Processor Time Memory % Committed Bytes In Use Memory Available Mbytes .NET CLR Exceptions # of Exceps Thrown/Sec .NET CLR Memory # Total Committed Bytes .NET CLR Memory % Time in GC ASP.NET Applications Requests in Application Queue ASP.NET Requests Queued ASP.NET Application Restarts

Component type	Workload type	Recommended metric
	Microsoft SQL Server Database Tier	CPUUtilization StatusCheckFailed Processor % Processor Time Memory % Committed Bytes In Use Memory Available Mbytes Paging File % Usage System Processor Queue Length Network Interface Bytes Total/Sec PhysicalDisk % Disk Time SQLServer:Buffer Manager Buffer Cache Hit ratio SQLServer:Buffer Manager Page Life Expectancy SQLServer:General Statistics Processes Blocked SQLServer:General Statistics User Connections SQLServer:Locks Number of Deadlocks/Sec SQLServer:SQL Statistics Batch Requests/Sec

Component type	Workload type	Recommended metric
	MySQL	CPUUtilization StatusCheckFailed Processor % Processor Time Memory % Committed Bytes In Use LogicalDisk % Free Space Memory Available Mbytes
	.NET workerpool/Mid-Tier	CPUUtilization StatusCheckFailed Processor % Processor Time Memory % Committed Bytes In Use Memory Available Mbytes .NET CLR Exceptions # of Exceps Thrown/Sec .NET CLR Memory # Total Committed Bytes .NET CLR Memory % Time in GC

Component type	Workload type	Recommended metric
	.NET Core Tier	CPUUtilization StatusCheckFailed Processor % Processor Time Memory % Committed Bytes In Use Memory Available Mbytes
	Oracle	CPUUtilization StatusCheckFailed Processor % Processor Time Memory % Committed Bytes In Use LogicalDisk % Free Space Memory Available Mbytes
	Postgres	CPUUtilization StatusCheckFailed Processor % Processor Time Memory % Committed Bytes In Use LogicalDisk % Free Space Memory Available Mbytes

Component type	Workload type	Recommended metric
	SharePoint	CPUUtilization StatusCheckFailed Processor % Processor Time Memory % Committed Bytes In Use Memory Available Mbytes ASP.NET Applications Cache API trims ASP.NET Requests Rejected ASP.NET Worker Process Restarts Memory Pages/sec SharePoint Publishing Cache Publishing cache flushes / second SharePoint Foundation Executing Time/Page Request SharePoint Disk-Based Cache Total number of cache compactions SharePoint Disk-Based Cache Blob cache hit ratio SharePoint Disk-Based Cache Blob Cache fill ratio

Component type	Workload type	Recommended metric
		SharePoint Disk-Based Cache Blob cache flushes / second ASP.NET Requests Queued ASP.NET Applications Requests in Application Queue ASP.NET Application Restarts LogicalDisk Avg. Disk sec/Write LogicalDisk Avg. Disk sec/Read Processor % Interrupt Time
EC2 instance (Linux servers)	Default/Custom	CPUUtilization StatusCheckFailed disk_used_percent mem_used_percent

Component type	Workload type	Recommended metric
	Java Application	CPUUtilization StatusCheckFailed disk_used_percent mem_used_percent java_lang_threading_threadcount java_lang_classloading_loadedclasscount java_lang_memory_heapmemoryusage_used java_lang_memory_heapmemoryusage_committed java_lang_operatingsystem_freephysicalmemorysize java_lang_operatingsystem_freeswapspacesize
	.NET Core Tier or SQL Server Database Tier	CPUUtilization StatusCheckFailed disk_used_percent mem_used_percent

Component type	Workload type	Recommended metric
	Oracle	CPUUtilization StatusCheckFailed disk_used_percent mem_used_percent
	Postgres	CPUUtilization StatusCheckFailed disk_used_percent mem_used_percent

Component type	Workload type	Recommended metric
EC2 instance group	SAP HANA multi-node or single node	• hanadb_server_startup_time_variations_seconds • hanadb_level_5_alerts_count • hanadb_level_4_alerts_count • hanadb_out_of_memory_events_count • hanadb_max_trigger_read_ratio_percent • hanadb_max_trigger_write_ratio_percent • hanadb_log_switch_race_ratio_percent • hanadb_time_since_last_savepoint_seconds • hanadb_disk_usage_highlevel_percent • hanadb_current_allocation_limit_used_percent • hanadb_table_allocation_limit_used_percent • hanadb_cpu_usage_percent • hanadb_plan_cache_hit_ratio_percent • hanadb_last_data_backup_age_days

Component type	Workload type	Recommended metric
EBS volume	Any	VolumeReadBytes VolumeWriteBytes VolumeReadOps VolumeWriteOps VolumeQueueLength VolumeThroughputPercentage VolumeConsumedReadWriteOps BurstBalance
Classic ELB	Any	HTTPCode_Backend_4XX HTTPCode_Backend_5XX Latency SurgeQueueLength UnHealthyHostCount
Application ELB	Any	HTTPCode_Target_4XX_Count HTTPCode_Target_5XX_Count TargetResponseTime UnHealthyHostCount

Component type	Workload type	Recommended metric
RDS Database instance	Any	CPUUtilization ReadLatency WriteLatency BurstBalance FailedSQLServerAgentJobsCount
RDS Database cluster	Any	CPUUtilization CommitLatency DatabaseConnections Deadlocks FreeableMemory NetworkThroughput VolumeBytesUsed
Lambda Function	Any	Duration Errors IteratorAge ProvisionedConcurrencySpilloverInvocations Throttles

Component type	Workload type	Recommended metric
SQS Queue	Any	ApproximateAgeOfOldestMessage ApproximateNumberOfMessagesVisible NumberOfMessagesSent
Amazon DynamoDB table	Any	SystemErrors UserErrors ConsumedReadCapacityUnits ConsumedWriteCapacityUnits ReadThrottleEvents WriteThrottleEvents ConditionalCheckFailedRequests TransactionConflict

Component type	Workload type	Recommended metric
Amazon S3 bucket	Any	If replication configuration with Replication Time Control (RTC) is enabled: ReplicationLatency BytesPendingReplication OperationsPendingReplicatio n If request metrics are turned on: 5xxErrors 4xxErrors BytesDownloaded BytesUploaded

Component type	Workload type	Recommended metric
Amazon Step Functions	Any	**General** • ExecutionThrottled • ExecutionsAborted • ProvisionedBucketSize • ProvisionedRefillRate • ConsumedCapacity **If state machine type is EXPRESS or log group level is OFF** • ExecutionsFailed • ExecutionsTimedOut **If state machine has Lambda functions** • LambdaFunctionsFailed • LambdaFunctionsTimedOut **If state machine has activitie s** • ActivitiesFailed • ActivitiesTimedOut • ActivitiesHeartbea tTimedOut **If state machine has service integrations** • ServiceIntegrationsFailed

Component type	Workload type	Recommended metric
		• ServiceIntegrationsTimedOut
API Gateway REST API stage	Any	• 4XXErrors • 5XXErrors • Latency

Component type	Workload type	Recommended metric
ECS Cluster	Any	CpuUtilized MemoryUtilized NetworkRxBytes NetworkTxBytes RunningTaskCount PendingTaskCount StorageReadBytes StorageWriteBytes CPUReservation (EC2 Launch Type only) CPUUtilization (EC2 Launch Type only) MemoryReservation (EC2 Launch Type only) MemoryUtilization (EC2 Launch Type only) GPUReservation (EC2 Launch Type only) instance_cpu_utilization (EC2 Launch Type only) instance_filesystem_utilization (EC2 Launch Type only) instance_memory_utilization (EC2 Launch Type only)

Component type	Workload type	Recommended metric
		instance_network_total_bytes (EC2 Launch Type only)

Component type	Workload type	Recommended metric
	Java Application	CpuUtilized MemoryUtilized NetworkRxBytes NetworkTxBytes RunningTaskCount PendingTaskCount StorageReadBytes StorageWriteBytes CPUReservation (EC2 Launch Type only) CPUUtilization (EC2 Launch Type only) MemoryReservation (EC2 Launch Type only) MemoryUtilization (EC2 Launch Type only) GPUReservation (EC2 Launch Type only) instance_cpu_utilization (EC2 Launch Type only) instance_filesystem_utiliza tion (EC2 Launch Type only) instance_memory_utilization (EC2 Launch Type only)

Component type	Workload type	Recommended metric
		instance_network_total_bytes (EC2 Launch Type only) java_lang_threading_threadcount java_lang_classloading_loadedclasscount java_lang_memory_heapmemoryusage_used java_lang_memory_heapmemoryusage_committed java_lang_operatingsystem_freephysicalmemorysize java_lang_operatingsystem_freeswapspacesize
ECS Service	Any	CPUUtilization MemoryUtilization CpuUtilized MemoryUtilized NetworkRxBytes NetworkTxBytes RunningTaskCount PendingTaskCount StorageReadBytes StorageWriteBytes

Component type	Workload type	Recommended metric
	Java Application	CPUUtilization MemoryUtilization CpuUtilized MemoryUtilized NetworkRxBytes NetworkTxBytes RunningTaskCount PendingTaskCount StorageReadBytes StorageWriteBytes java_lang_threading_threadcount java_lang_classloading_loadedclasscount java_lang_memory_heapmemoryusage_used java_lang_memory_heapmemoryusage_committed java_lang_operatingsystem_freephysicalmemorysize java_lang_operatingsystem_freeswapspacesize

Component type	Workload type	Recommended metric
EKS Cluster	Any	cluster_failed_node_count node_cpu_reserved_capacity node_cpu_utilization node_filesystem_utilization node_memory_reserved_capacity node_memory_utilization node_network_total_bytes pod_cpu_reserved_capacity pod_cpu_utilization pod_cpu_utilization_over_pod_limit pod_memory_reserved_capacity pod_memory_utilization pod_memory_utilization_over_pod_limit pod_network_rx_bytes pod_network_tx_bytes

Component type	Workload type	Recommended metric
	Java Application	cluster_failed_node_count node_cpu_reserved_capacity node_cpu_utilization node_filesystem_utilization node_memory_reserved_capacity node_memory_utilization node_network_total_bytes pod_cpu_reserved_capacity pod_cpu_utilization pod_cpu_utilization_over_pod_limit pod_memory_reserved_capacity pod_memory_utilization pod_memory_utilization_over_pod_limit pod_network_rx_bytes pod_network_tx_bytes java_lang_threading_threadcount java_lang_classloading_loadedclasscount

Component type	Workload type	Recommended metric
		java_lang_memory_heapmemoryusage_used java_lang_memory_heapmemoryusage_committed java_lang_operatingsystem_freephysicalmemorysize java_lang_operatingsystem_freeswapspacesize

Component type	Workload type	Recommended metric
Kubernetes Cluster on EC2	Any	cluster_failed_node_count node_cpu_reserved_capacity node_cpu_utilization node_filesystem_utilization node_memory_reserved_capacity node_memory_utilization node_network_total_bytes pod_cpu_reserved_capacity pod_cpu_utilization pod_cpu_utilization_over_pod_limit pod_memory_reserved_capacity pod_memory_utilization pod_memory_utilization_over_pod_limit pod_network_rx_bytes pod_network_tx_bytes

Component type	Workload type	Recommended metric
	Java Application	cluster_failed_node_count node_cpu_reserved_capacity node_cpu_utilization node_filesystem_utilization node_memory_reserved_capacity node_memory_utilization node_network_total_bytes pod_cpu_reserved_capacity pod_cpu_utilization pod_cpu_utilization_over_pod_limit pod_memory_reserved_capacity pod_memory_utilization pod_memory_utilization_over_pod_limit pod_network_rx_bytes pod_network_tx_bytes java_lang_threading_threadcount java_lang_classloading_loadedclasscount

Component type	Workload type	Recommended metric
		java_lang_memory_heapmemoryusage_used java_lang_memory_heapmemoryusage_committed java_lang_operatingsystem_freephysicalmemorysize java_lang_operatingsystem_freeswapspacesize

The following table lists the recommended processes and process metrics for each component type. CloudWatch Application Insights does not recommend process monitoring for processes that do not run on an instance.

Component type	Workload type	Recommended process	Recommended metric
EC2 instance (Windows servers)	Microsoft IIS/.NET Web Front-End	`w3wp`	`procstat cpu_usage` , `procstat memory_rss` , `procstat memory_vms` , `procstat read_bytes` , `procstat write_bytes`
	Microsoft SQL Server Database Tier	`SQLAgent`	`procstat cpu_usage` ,

Component type	Workload type	Recommended process	Recommended metric
			procstat memory_rss , procstat memory_vms , procstat read_bytes , procstat write_bytes
		sqlservr	procstat cpu_usage , procstat memory_rss , procstat memory_vms , procstat read_bytes , procstat write_bytes
		sqlwriter	procstat cpu_usage , procstat memory_rss

Component type	Workload type	Recommended process	Recommended metric
		Reporting ServicesS ervice	procstat cpu_usage , procstat memory_rss
		MsDtsServr	procstat cpu_usage , procstat memory_rss , procstat memory_vms , procstat read_bytes , procstat write_bytes
		Msmdsrv	procstat cpu_usage , procstat memory_rss , procstat memory_vms , procstat read_bytes , procstat write_bytes

Component type	Workload type	Recommended process	Recommended metric
	.NET workerpool/ Mid-Tier	w3wp	procstat cpu_usage , procstat memory_rss , procstat memory_vms , procstat read_bytes , procstat write_bytes
	.NET Core Tier	w3wp	procstat cpu_usage , procstat memory_rss , procstat memory_vms , procstat read_bytes , procstat write_bytes

Performance Counter metrics

Performance Counter metrics are recommended for instances only when the corresponding Performance Counter sets are installed on the Windows instances.

Performance Counter metric name	Performance Counter set name
.NET CLR Exceptions # of Exceps Thrown	.NET CLR Exceptions
.NET CLR Exceptions # of Exceps Thrown/Sec	.NET CLR Exceptions
.NET CLR Exceptions # of Filters/Sec	.NET CLR Exceptions
.NET CLR Exceptions # of Finallys/Sec	.NET CLR Exceptions
.NET CLR Exceptions Throw to Catch Depth/ Sec	.NET CLR Exceptions
.NET CLR Interop # of CCWs	.NET CLR Interop
.NET CLR Interop # of Stubs	.NET CLR Interop
.NET CLR Interop # of TLB exports/Sec	.NET CLR Interop
.NET CLR Interop # of TLB imports/Sec	.NET CLR Interop
.NET CLR Interop # of Marshaling	.NET CLR Interop
.NET CLR Jit % Time in Jit	.NET CLR Jit
.NET CLR Jit Standard Jit Failures	.NET CLR Jit
.NET CLR Loading % Time Loading	.NET CLR Loading
.NET CLR Loading Rate of Load Failures	.NET CLR Loading
.NET CLR LocksAndThreads Contention Rate/ Sec	.NET CLR LocksAndThreads
.NET CLR LocksAndThreads Queue Length/Sec	.NET CLR LocksAndThreads
.NET CLR Memory # Total Committed Bytes	.NET CLR Memory
.NET CLR Memory % Time in GC	.NET CLR Memory

Performance Counter metric name	Performance Counter set name
.NET CLR Networking 4.0.0.0 HttpWebRequest Average Queue Time	.NET CLR Networking 4.0.0.0
.NET CLR Networking 4.0.0.0 HttpWebRe quests Aborted/Sec	.NET CLR Networking 4.0.0.0
.NET CLR Networking 4.0.0.0 HttpWebRe quests Failed/Sec	.NET CLR Networking 4.0.0.0
.NET CLR Networking 4.0.0.0 HttpWebRe quests Queued/Sec	.NET CLR Networking 4.0.0.0
APP_POOL_WAS Total Worker Process Ping Failures	APP_POOL_WAS
ASP.NET Application Restarts	ASP.NET
ASP.NET Requests Rejected	ASP.NET
ASP.NET Worker Process Restarts	ASP.NET
ASP.NET Applications Cache API trims	ASP.NET Applications
ASP.NET Applications % Managed Processor Time (estimated)	ASP.NET Applications
ASP.NET Applications Errors Total/Sec	ASP.NET Applications
ASP.NET Applications Errors Unhandled During Execution/Sec	ASP.NET Applications
ASP.NET Applications Requests in Application Queue	ASP.NET Applications
ASP.NET Applications Requests/Sec	ASP.NET Applications
ASP.NET Request Wait Time	ASP.NET
ASP.NET Requests Queued	ASP.NET

Performance Counter metric name	Performance Counter set name
Database ==> Instances Database Cache % Hit	Database ==> Instances
Database ==> Instances I/O Database Reads Average Latency	Database ==> Instances
Database ==> Instances I/O Database Reads/ sec	Database ==> Instances
Database ==> Instances I/O Log Writes Average Latency	Database ==> Instances
DirectoryServices DRA Pending Replication Operations	DirectoryServices
DirectoryServices DRA Pending Replication Synchronizations	DirectoryServices
DirectoryServices LDAP Bind Time	DirectoryServices
DNS Recursive Queries/sec	DNS
DNS Recursive Query Failure/sec	DNS
DNS TCP Query Received/sec	DNS
DNS Total Query Received/sec	DNS
DNS Total Response Sent/sec	DNS
DNS UDP Query Received/sec	DNS
HTTP Service Request Queues CurrentQu eueSize	HTTP Service Request Queues
LogicalDisk % Free Space	LogicalDisk
LogicalDisk Avg. Disk sec/Write	LogicalDisk
LogicalDisk Avg. Disk sec/Read	LogicalDisk

Performance Counter metric name	Performance Counter set name
LogicalDisk Avg. Disk Queue Length	LogicalDisk
Memory % Committed Bytes In Use	Memory
Memory Available Mbytes	Memory
Memory Pages/Sec	Memory
Memory Long-Term Average Standby Cache Lifetime (s)	Memory
Network Interface Bytes Total/Sec	Network Interface
Network Interface Bytes Received/sec	Network Interface
Network Interface Bytes Sent/sec	Network Interface
Network Interface Current Bandwidth	Network Interface
Paging File % Usage	Paging File
PhysicalDisk % Disk Time	PhysicalDisk
PhysicalDisk Avg. Disk Queue Length	PhysicalDisk
PhysicalDisk Avg. Disk Sec/Read	PhysicalDisk
PhysicalDisk Avg. Disk Sec/Write	PhysicalDisk
PhysicalDisk Disk Read Bytes/Sec	PhysicalDisk
PhysicalDisk Disk Reads/Sec	PhysicalDisk
PhysicalDisk Disk Write Bytes/Sec	PhysicalDisk
PhysicalDisk Disk Writes/Sec	PhysicalDisk
Processor % Idle Time	Processor
Processor % Interrupt Time	Processor

Performance Counter metric name	Performance Counter set name
Processor % Processor Time	Processor
Processor % User Time	Processor
SharePoint Disk-Based Cache Blob Cache fill ratio	SharePoint Disk-Based Cache
SharePoint Disk-Based Cache Blob cache flushes / second	SharePoint Disk-Based Cache
SharePoint Disk-Based Cache Blob cache hit ratio	SharePoint Disk-Based Cache
SharePoint Disk-Based Cache Total number of cache compactions	SharePoint Disk-Based Cache
SharePoint Foundation Executing Time/Page Request	SharePoint Foundation
SharePoint Publishing Cache Publishing cache flushes / second	SharePoint Publishing Cache
Security System-Wide Statistics Kerberos Authentications	Security System-Wide Statistics
Security System-Wide Statistics NTLM Authentications	Security System-Wide Statistics
SQLServer:Access Methods Forwarded Records/Sec	SQLServer:Access Methods
SQLServer:Access Methods Full Scans/Sec	SQLServer:Access Methods
SQLServer:Access Methods Page Splits/Sec	SQLServer:Access Methods
SQLServer:Buffer Manager Buffer cache hit Ratio	SQLServer:Buffer Manager

Performance Counter metric name	Performance Counter set name
SQLServer:Buffer Manager Page life Expectanc y	SQLServer:Buffer Manager
SQLServer:Database Replica File Bytes Received/sec	SQLServer:Database Replica
SQLServer:Database Replica Log Bytes Received/sec	SQLServer:Database Replica
SQLServer:Database Replica Log remaining for undo	SQLServer:Database Replica
SQLServer:Database Replica Log Send Queue	SQLServer:Database Replica
SQLServer:Database Replica Mirrored Write Transaction/sec	SQLServer:Database Replica
SQLServer:Database Replica Recovery Queue	SQLServer:Database Replica
SQLServer:Database Replica Redo Bytes Remaining	SQLServer:Database Replica
SQLServer:Database Replica Redone Bytes/sec	SQLServer:Database Replica
SQLServer:Database Replica Total Log requiring undo	SQLServer:Database Replica
SQLServer:Database Replica Transaction Delay	SQLServer:Database Replica
SQLServer:General Statistics Processes Blocked	SQLServer:General Statistics
SQLServer:General Statistics User Connections	SQLServer:General Statistics
SQLServer:Latches Average Latch Wait Time (ms)	SQLServer:Latches
SQLServer:Locks Average Wait Time (ms)	SQLServer:Locks

Performance Counter metric name	Performance Counter set name
SQLServer:Locks Lock Timeouts/Sec	SQLServer:Locks
SQLServer:Locks Lock Waits/Sec	SQLServer:Locks
SQLServer:Locks Number of Deadlocks/Sec	SQLServer:Locks
SQLServer:Memory Manager Memory Grants Pending	SQLServer:Memory Manager
SQLServer:SQL Statistics Batch Requests/Sec	SQLServer:SQL Statistics
SQLServer:SQL Statistics SQL Compilations/ Sec	SQLServer:SQL Statistics
SQLServer:SQL Statistics SQL Re-Compil ations/Sec	SQLServer:SQL Statistics
System Processor Queue Length	System
TCPv4 Connections Established	TCPv4
TCPv6 Connections Established	TCPv6
W3SVC_W3WP File Cache Flushes	W3SVC_W3WP
W3SVC_W3WP File Cache Misses	W3SVC_W3WP
W3SVC_W3WP Requests/Sec	W3SVC_W3WP
W3SVC_W3WP URI Cache Flushes	W3SVC_W3WP
W3SVC_W3WP URI Cache Misses	W3SVC_W3WP
Web Service Bytes Received/Sec	Web Service
Web Service Bytes Sent/Sec	Web Service
Web Service Connection Attempts/Sec	Web Service
Web Service Current Connections	Web Service

Performance Counter metric name	Performance Counter set name
Web Service Get Requests/Sec	Web Service
Web Service Post Requests/Sec	Web Service

Using the resource health view in the CloudWatch console

You can use the resource health view to automatically discover, manage, and visualize the health and performance of hosts across their applications in a single view. You can visualize the health of their hosts by a performance dimension such as CPU or memory, and slice and dice hundreds of hosts in a single view using filters. You can filter by tags or by use cases, such as hosts in the same Auto Scaling group or hosts that use the same load balancer,

Prerequisites

To make sure that you get the full benefit of the resource health view, check that you have the following prerequisites.

- To see the memory utilization of your hosts and use it as a filter, you must install the CloudWatch agent on your hosts and set it up to send a memory metric to CloudWatch in the default `CWAgent` namespace. On Linux and macOS instances, the CloudWatch agent must send the `mem_used_percent` metric. On Windows instances, the agent must send the `Memory % Committed Bytes In Use` metric. These metrics are included if you use the wizard to create the CloudWatch agent configuration file and select any of the pre-defined sets of metrics. Metrics collected by the CloudWatch agent are billed as custom metrics. For more information, see Install the CloudWatch agent.

  When you use the CloudWatch agent to collect these memory metrics to use with the resource health view, you must include the following section in the CloudWatch agent configuration file. This section contains the default dimension settings and is created by default, so do not change any part of this section to anything different than what is shown in the following example.

```
"append_dimensions": {
  "ImageId": "${aws:ImageId}",
  "InstanceId": "${aws:InstanceId}",
  "InstanceType": "${aws:InstanceType}",
  "AutoScalingGroupName": "${aws:AutoScalingGroupName}"
```

```
},
```

- To view all the information available in the resource health view, you must be signed in to an account that has the following permissions. If you are signed on with fewer permissions, you can still use the resource health view but some performance data will not be visible.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "autoscaling:Describe*",
                "cloudwatch:Describe*",
                "cloudwatch:Describe*",
                "cloudwatch:Get*",
                "cloudwatch:List*",
                "logs:Get*",
                "logs:Describe*",
                "sns:Get*",
                "sns:List*",
                "ec2:DescribeInstances",
                "ec2:DescribeInstanceStatus",
                "ec2:DescribeRegions"
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
    ]
}
```

To view resource health in your account

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Insights**, **EC2 Resource Health**.

   The resource health page appears, showing a square for each host in your account. Each square is colored based on the current status of that host, based on the setting for **Color by**. Host squares with an alarm symbol have one or more alarms currently in ALARM state.

   You can see up to 500 hosts in a single view. If you have more hosts in your account, use the filter settings in step 6 of this procedure.

3. To change what criteria is used to show each host's health, choose a setting for **Color by**. You can choose **CPU Utilization**, **Memory Utilization**, or **Status check**. Memory utilization metrics are available only for hosts that are running the CloudWatch agent and have it configured to collect memory metrics and send them to the default `CWAgent` namespace. For more information, see Collect metrics, logs, and traces with the CloudWatch agent.
4. To change the thresholds and the colors that are used for the health indicators in the grid, choose the gear icon above the grid.
5. To toggle whether to show alarms in the host grid, choose or clear **Show alarms across all metrics**.
6. To split the hosts in the map into groups, choose a grouping criteria for **Group by**.
7. To narrow the view to fewer hosts, choose a filter criteria for **Filter by**. You can filter by tags and by resource groupings such as Auto Scaling group, instance type, security group, and more.
8. To sort hosts, choose a sorting criteria for **Sort by**. You can sort by status check results, instance state, CPU or memory utilization, and the number of alarms that are in ALARM state.
9. To see more information about a host, choose the square that represents that host. A popup pane appears. To then dive deeper into information about that host, choose **View dashboard** or **View on list**.

Application performance monitoring (APM)

CloudWatch Application Signals provides application performance monitoring (APM) features such as pre-built, standardized dashboards for critical application metrics, correlated trace spans, and a service map to enable you to visualize interactions between applications and their dependencies. You can also search and analyze transaction spans and trace summaries to debug distributed application issues in a business context, for cases such as troubleshooting customer support tickets or finding top impacted customers. You can also create Service Level Objectives (SLOs) to closely track the performance KPIs of critical operations in your application, enabling you to easily identify and triage operations that do not meet your business KPIs.

See the following sections for an overview of these troubleshooting capabilities:

- Monitor the operational health of your applications with Application Signals
- Searching and analyzing spans

Start collecting application metrics and traces

Get the most integrated application performance monitoring experience by auto-instrumenting applications to easily collect telemetry, whether they are running in Amazon EKS clusters, Amazon EC2, Amazon ECS, Kubernetes, Lambda, or on-premise. Optionally, you can also use OpenTelemetry with Application Signals to collect telemetry.

> **Note**
>
> You must enable transaction search to get all APM features along with a new unified pricing for CloudWatch Application Signals, inclusive of X-Ray traces and application transaction spans. For more information about pricing, see Amazon CloudWatch Pricing.

Topics

- Application Signals
- Service level objectives (SLOs)
- Transaction Search
- Synthetic monitoring (canaries)
- CloudWatch RUM

- Perform launches and A/B experiments with CloudWatch Evidently

Application Signals

Use CloudWatch Application Signals to automatically instrument your applications on Amazon so that you can monitor current application health and track long-term application performance against your business objectives. Application Signals provides you with a unified, application-centric view of your applications, services, and dependencies, and helps you monitor and triage application health.

- Enable Application Signals to automatically collect metrics and traces from your applications, and display key metrics such as call volume, availability, latency, faults, and errors. Quickly see and triage current operational health, and whether your applications are meeting their longer-term performance goals, without writing custom code or creating dashboards.
- Create and monitor service-level objectives (SLOs) with Application Signals. Easily create and track status of SLOs related to CloudWatch metrics, including the new standard application metrics that Application Signals collects. See and track the service level indicator (SLI) status of your application services within a services list and topology map. Create alarms to track your SLOs, and track the new standard application metrics that Application Signals collects.
- See a map of your application topology that Application Signals automatically discovers, that gives you a visual representation of your applications, dependencies, and their connectivity.
- Application Signals works with CloudWatch RUM, CloudWatch Synthetics canaries, Amazon Service Catalog AppRegistry, and Amazon EC2 Auto Scaling to display your client pages, Synthetics canaries, and application names within dashboards and maps.

Topics

- Enable your applications on Kubernetes
- Enable your applications on Lambda
- Troubleshooting your Application Signals installation
- (Optional) Configuring Application Signals
- Monitor the operational health of your applications with Application Signals
- Metrics collected by Application Signals

Use Application Signals for daily application monitoring

Use Application Signals within the CloudWatch console, as part of daily application monitoring:

1. If you have created service level objectives (SLOs) for your services, start with the Service Level Objectives (SLO) page. This gives you an immediate view of the health of your most critical services, operations, and dependencies. Choose the service, operation, or dependency name for an SLO to open the Service detail page and see detailed service information as you troubleshoot issues.
2. Open the Services page to see a summary of all your services, and quickly see services with the highest fault rate or latency. If you have created SLOs, look at the Services table to see which services have unhealthy service level indicators (SLIs). If a particular service is in an unhealthy state, select the service to open the Service detail page and see service operations, dependencies, Synthetics canaries, and client requests. Select a point in a graph to see correlated traces so that you can troubleshoot and identify the root cause of operational issues.
3. If new services have been deployed or dependencies have changed, open the Service Map to inspect your application topology. See a map of your applications that shows the relationship between clients, Synthetics canaries, services, and dependencies. Quickly see SLI health, view key metrics such as call volume, fault rate, and latency, and drill down to see more detailed information in the Service detail page.

Using Application Signals incurs charges. For information about CloudWatch pricing, see Amazon CloudWatch Pricing.

> **Note**
>
> It is not necessary to enable Application Signals to use CloudWatch Synthetics, CloudWatch RUM, or CloudWatch Evidently. However, Synthetics and CloudWatch RUM work with Application Signals to provide benefits when you use these features together.

Application Signals cross-account

With Application Signals cross-account observability, you can monitor and troubleshoot your applications that span multiple Amazon accounts within a single Region.

You can use Amazon CloudWatch Observability Access Manager to set up one or more of your Amazon accounts as a monitoring account. You'll provide the monitoring account with the ability to view data in your source account by creating a sink in your monitoring account. You use the sink to create a link from your source account to your monitoring account. For more information, see CloudWatch cross-account observability.

Required resources

For proper functionality of Application Signals cross-account observability, ensure that the following telemetry types are shared through the CloudWatch Observability Access Manager.

- Application Signals services and service level objectives (SLOs)
- Metrics in Amazon CloudWatch
- Log groups in Amazon CloudWatch Logs
- Traces in Amazon X-Ray

Supported languages and architectures

Application Signals supports Java, Python, Node.js, and .NET applications.

Application Signals is supported and tested on Amazon EKS, Amazon ECS, and Amazon EC2. On Amazon EKS clusters, it automatically discovers the names of your services and clusters. On other architectures, you must supply the names of services and environments when you enable those services for Application Signals.

The instructions for enabling Application Signals on Amazon EC2 should work on any architecture that supports the CloudWatch agent and Amazon Distro for OpenTelemetry. However, the instructions have not been tested on architectures other than Amazon ECS and Amazon EC2.

Supported Regions

Application Signals is supported in every commercial Region except for Canada West (Calgary).

Permissions required for Application Signals

This section explains the permissions necessary for you to enable, manage, and operate Application Signals.

Permissions to enable and manage Application Signals

To manage Application Signals, you must be signed on with the following permissions:

```
{
 "Version": "2012-10-17",
 "Statement": [
    {
        "Sid": "CloudWatchApplicationSignalsFullAccessPermissions",
        "Effect": "Allow",
        "Action": "application-signals:*",
        "Resource": "*"
    },
    {
        "Sid": "CloudWatchApplicationSignalsAlarmsPermissions",
        "Effect": "Allow",
        "Action": [
            "cloudwatch:DescribeAlarms"
        ],
        "Resource": "*"
    },
    {
        "Sid": "CloudWatchApplicationSignalsMetricsPermissions",
        "Effect": "Allow",
        "Action": [
            "cloudwatch:GetMetricData",
            "cloudwatch:ListMetrics"
        ],
        "Resource": "*"
    },
```

```
    {
        "Sid": "CloudWatchApplicationSignalsLogGroupPermissions",
        "Effect": "Allow",
        "Action": [
            "logs:StartQuery",
            "logs:DescribeMetricFilters"
        ],
        "Resource": "arn:aws:logs:*:*:log-group:/aws/application-signals/data:*"
    },
    {
        "Sid": "CloudWatchApplicationSignalsLogsPermissions",
        "Effect": "Allow",
        "Action": [
            "logs:GetQueryResults",
            "logs:StopQuery"
        ],
        "Resource": "*"
    },
    {
        "Sid": "CloudWatchApplicationSignalsSyntheticsPermissions",
        "Effect": "Allow",
        "Action": [
            "synthetics:DescribeCanaries",
            "synthetics:DescribeCanariesLastRun",
            "synthetics:GetCanaryRuns"
        ],
        "Resource": "*"
    },
    {
        "Sid": "CloudWatchApplicationSignalsRumPermissions",
        "Effect": "Allow",
        "Action": [
            "rum:BatchCreateRumMetricDefinitions",
            "rum:BatchDeleteRumMetricDefinitions",
            "rum:BatchGetRumMetricDefinitions",
            "rum:GetAppMonitor",
            "rum:GetAppMonitorData",
            "rum:ListAppMonitors",
            "rum:PutRumMetricsDestination",
            "rum:UpdateRumMetricDefinition"
        ],
        "Resource": "*"
    },
    {
```

```
        "Sid": "CloudWatchApplicationSignalsXrayPermissions",
        "Effect": "Allow",
        "Action": [
            "xray:GetTraceSummaries"
        ],
        "Resource": "*"
    },
    {
        "Sid": "CloudWatchApplicationSignalsPutMetricAlarmPermissions",
        "Effect": "Allow",
        "Action": "cloudwatch:PutMetricAlarm",
        "Resource": [
            "arn:aws:cloudwatch:*:*:alarm:SLO-AttainmentGoalAlarm-*",
            "arn:aws:cloudwatch:*:*:alarm:SLO-WarningAlarm-*",
            "arn:aws:cloudwatch:*:*:alarm:SLI-HealthAlarm-*"
        ]
    },
    {
        "Sid": "CloudWatchApplicationSignalsCreateServiceLinkedRolePermissions",
        "Effect": "Allow",
        "Action": "iam:CreateServiceLinkedRole",
        "Resource": "arn:aws:iam::*:role/aws-service-role/application-
signals.cloudwatch.amazonaws.com/AWSServiceRoleForCloudWatchApplicationSignals",
        "Condition": {
            "StringLike": {
                "iam:AWSServiceName": "application-signals.cloudwatch.amazonaws.com"
            }
        }
    },
    {
        "Sid": "CloudWatchApplicationSignalsGetRolePermissions",
        "Effect": "Allow",
        "Action": "iam:GetRole",
        "Resource": "arn:aws:iam::*:role/aws-service-role/application-
signals.cloudwatch.amazonaws.com/AWSServiceRoleForCloudWatchApplicationSignals"
    },
    {
        "Sid": "CloudWatchApplicationSignalsSnsWritePermissions",
        "Effect": "Allow",
        "Action": [
            "sns:CreateTopic",
            "sns:Subscribe"
        ],
        "Resource": "arn:aws:sns:*:*:cloudwatch-application-signals-*"
```

```
    },
    {
        "Sid": "CloudWatchApplicationSignalsSnsReadPermissions",
        "Effect": "Allow",
        "Action": "sns:ListTopics",
        "Resource": "*"
    }
  ]
}
```

To enable Application Signals on Amazon EC2, or custom architectures, see Enable Application Signals on Amazon EC2. To enable and manage Application Signals on Amazon EKS using the Amazon CloudWatch Observability EKS add-on, you need the following permissions.

> **Important**
>
> These permissions include `iam:PassRole` with `Resource "*"` and `eks:CreateAddon` with `Resource "*"`. These are powerful permissions and you should use caution in granting them.

```
{
"Version": "2012-10-17",
    "Statement": [
        {
"Sid": "CloudWatchApplicationSignalsEksAddonManagementPermissions",
            "Effect": "Allow",
            "Action": [
                "eks:AccessKubernetesApi",
                "eks:CreateAddon",
                "eks:DescribeAddon",
                "eks:DescribeAddonConfiguration",
                "eks:DescribeAddonVersions",
                "eks:DescribeCluster",
                "eks:DescribeUpdate",
                "eks:ListAddons",
                "eks:ListClusters",
                "eks:ListUpdates",
                "iam:ListRoles",
                "iam:PassRole"
            ],
            "Resource": "*"
```

```
        },
        {
"Sid":
 "CloudWatchApplicationSignalsEksCloudWatchObservabilityAddonManagementPermissions",
            "Effect": "Allow",
            "Action": [
                "eks:DeleteAddon",
                "eks:UpdateAddon"
            ],
            "Resource": "arn:aws:eks:*:*:addon/*/amazon-cloudwatch-observability/*"
        }
    ]
}
```

The Application Signals dashboard shows the Amazon Service Catalog AppRegistry applications that your SLOs are associated with. To see these applications in the SLO pages, you must have the following permissions:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CloudWatchApplicationSignalsTaggingReadPermissions",
            "Effect": "Allow",
            "Action": "tag:GetResources",
            "Resource": "*"
        }
    ]
}
```

Operating Application Signals

Service operators who are using Application Signals to monitor services and SLOs must be signed on to an account with the following read only permissions:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CloudWatchApplicationSignalsReadOnlyAccessPermissions",
            "Effect": "Allow",
            "Action": [
```

```
                "application-signals:BatchGet*",
                "application-signals:Get*",
                "application-signals:List*"
            ],
            "Resource": "*"
        },
        {
            "Sid": "CloudWatchApplicationSignalsGetRolePermissions",
            "Effect": "Allow",
            "Action": "iam:GetRole",
            "Resource": "arn:aws:iam::*:role/aws-service-role/application-
signals.cloudwatch.amazonaws.com/AWSServiceRoleForCloudWatchApplicationSignals"
        },
        {
        "Sid": "CloudWatchApplicationSignalsLogGroupPermissions",
        "Effect": "Allow",
        "Action": [
            "logs:StartQuery",
            "logs:DescribeMetricFilters"
        ],
        "Resource": "arn:aws:logs:*:*:log-group:/aws/application-signals/data:*"
    },
    {
        "Sid": "CloudWatchApplicationSignalsLogsPermissions",
        "Effect": "Allow",
        "Action": [
            "logs:GetQueryResults",
            "logs:StopQuery"
        ],
        "Resource": "*"
    },
        {
            "Sid": "CloudWatchApplicationSignalsAlarmsReadPermissions",
            "Effect": "Allow",
            "Action": [
                "cloudwatch:DescribeAlarms"
            ],
            "Resource": "*"
        },
        {
            "Sid": "CloudWatchApplicationSignalsMetricsReadPermissions",
            "Effect": "Allow",
            "Action": [
                "cloudwatch:GetMetricData",
```

```
                "cloudwatch:ListMetrics"
            ],
            "Resource": "*"
        },
        {
            "Sid": "CloudWatchApplicationSignalsSyntheticsReadPermissions",
            "Effect": "Allow",
            "Action": [
                "synthetics:DescribeCanaries",
                "synthetics:DescribeCanariesLastRun",
                "synthetics:GetCanaryRuns"
            ],
            "Resource": "*"
        },
        {
            "Sid": "CloudWatchApplicationSignalsRumReadPermissions",
            "Effect": "Allow",
            "Action": [
                "rum:BatchGetRumMetricDefinitions",
                "rum:GetAppMonitor",
                "rum:GetAppMonitorData",
                "rum:ListAppMonitors"
            ],
            "Resource": "*"
        },
        {
            "Sid": "CloudWatchApplicationSignalsXrayReadPermissions",
            "Effect": "Allow",
            "Action": [
                "xray:GetTraceSummaries"
            ],
            "Resource": "*"
        }
    ]
}
```

To see which Amazon Service Catalog AppRegistry Applications that your SLOs are associated within the Application Signals dashboard, you must have the following permissions:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
```

```
            "Sid": "CloudWatchApplicationSignalsTaggingReadPermissions",
            "Effect": "Allow",
            "Action": "tag:GetResources",
            "Resource": "*"
        }
    ]
}
```

To check if Application Signals on Amazon EKS using the [Amazon CloudWatch Observability EKS add-on](#) is enabled, you need to have the following permissions:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CloudWatchApplicationSignalsEksReadPermissions",
            "Effect": "Allow",
            "Action": [
                "eks:ListAddons",
                "eks:ListClusters"
            ],
            "Resource": "*"
        },
        {
            "Sid": "CloudWatchApplicationSignalsEksDescribeAddonReadPermissions",
            "Effect": "Allow",
            "Action": [
                "eks:DescribeAddon"
            ],
            "Resource": "arn:aws:eks:*:*:addon/*/amazon-cloudwatch-observability/*"
        }
    ]
}
```

Supported systems

Application Signals is supported and tested on Amazon EKS, native Kubernetes, Amazon ECS, and Amazon EC2. The instructions for enabling Application Signals on Amazon EC2 should work on any platform that supports the CloudWatch agent and Amazon Distro for OpenTelemetry, but the instructions have not been tested on other platforms.

Topics

Java compatibility

Application Signals supports Java applications, and supports the same Java libraries and frameworks as the Amazon Distro for OpenTelemetry does. For more information, see Supported libraries, frameworks, application servers, and JVMs.

JVM versions 8, 11, 17, 21, and 23 are supported.

Python compatibility

Python compatibility

Application Signals supports the same libraries and frameworks as the Amazon Distro for OpenTelemetry does. For more information, see **Supported packages** at opentelemetry-python-contrib.

Python versions 3.8 and later are supported.

Before you enable Application Signals for your Python applications, be aware of the following considerations.

- In some containerized applications, a missing `PYTHONPATH` environment variable can sometimes cause the application to fail to start. To resolve this, ensure that you set the `PYTHONPATH` environment variable to the location of your application's working directory. This is due to a known issue with OpenTelemetry auto-instrumentation. For more information about this issue, see Python autoinstrumentation setting of PYTHONPATH is not compliant.
- For Django applications, there are additional required configurations, which are outlined in the OpenTelemetry Python documentation.
  - Use the `--noreload` flag to prevent automatic reloading.

- Set the `DJANGO_SETTINGS_MODULE` environment variable to the location of your Django application's `settings.py` file. This ensures that OpenTelemetry can correctly access and integrate with your Django settings.

.NET compatibility

Application Signals supports .NET applications with Amazon Distro for Open Telemetry (ADOT) instrumentation on Amazon EKS, Amazon EC2, Amazon ECS and Kubernetes running on Amazon EC2.

This release supports .NET 6 and 8, and .NET Framework 4.6.2 and higher.

Application Signals supports .NET applications that are running on x86-64 or ARM64 CPUs, and supports the Linux x64, Linux ARM64, Microsoft Windows Server 2022 x64, and Microsoft Windows Server 2019 x64 operating systems.

Node.js compatibility

Application Signals supports the same Node.js libraries and frameworks as the Amazon Distro for OpenTelemetry does. For more information, see Supported instrumentations.

This release supports Node.js versions 14, 16, 18, 20, and 22.

Known limitations about Node.js with ESM

The Amazon Distro for Opentelemetry Node.js supports two module systems: ECMAScript Modules (ESM) and CommonJS (CJS). To enable Application Signals, we recommend that you use the CJS module format because OpenTelemetry JavaScript's support of ESM is experimental and a work in progress. For more details, see ECMAScript Modules vs. CommonJS on GitHub.

To determine if your application is using CJS and not ESM, ensure that your application does not fulfill the conditions to enable ESM. For more information about these conditions, see Enabling in the Node.js documentation.

The Amazon Distro for Opentelemetry Node.js provides limited support for ESM based on OpenTelemetry JavaScript's experimental support for ESM. This means the following:

- The Node.js version must be 18.19.0 or later.

- The Node.js application that you want to instrument must include `@aws/aws-distro-opentelemetry-node-autoinstrumentation` and `@opentelemetry/instrumentation` as dependencies.
- The Node.js application that you want to instrument must start with the following node option:

```
NODE_OPTIONS=' --import @aws/aws-distro-opentelemetry-node-autoinstrumentation/
register --experimental-loader=@opentelemetry/instrumentation/hook.mjs'
```

To enable Application Signals with Node.js ESM module format, we provide the different setup for different platforms:

- **Amazon EKS** – the section called “Setting up a Node.js application with the ESM module format”
- **Amazon ECS with sidecar strategy** – Setting up a Node.js application with the ESM module format
- **Amazon ECS with daemon strategy** – Setting up a Node.js application with the ESM module format
- **Amazon ECS with Amazon CDK**
- **Amazon EC2** – Setting up a Node.js application with the ESM module format
- **Kubernetes** – the section called “Setting up a Node.js application with the ESM module format”

OpenTelemetry compatibility

CloudWatch Application Signals is fully compatible with OpenTelemetry. For information about how to get started, see OpenTelemetry with CloudWatch. If you want a more integrated experience, such as using the CloudWatch Agent with Amazon Distro for OpenTelemetry (ADOT) SDKs, see Getting started with Application Signals to find your preferred configuration method.

Known issues

The runtime metrics collection in the Java SDK release v1.32.5 is known to not work with applications using JBoss Wildfly. This issue extends to the Amazon CloudWatch Observability EKS add-on, affecting versions `2.3.0-eksbuild.1` through `2.6.0-eksbuild.1`. The issue is fixed in Java SDK release `v1.32.6` and the Amazon CloudWatch Observability EKS add-on version `v3.0.0-eksbuild.1`.

If you are impacted, either upgrade the Java SDK version or disable your runtime metrics collection by adding the environment variable `OTEL_AWS_APPLICATION_SIGNALS_RUNTIME_ENABLED=false` to your application.

Supported instrumentation setups

You can enable CloudWatch Application Signals with different instrumentation setups. This topic describes each of the setup methods and recommendations based on the method you choose.

Use Amazon Distro for OpenTelemetry with the CloudWatch Agent

The most integrated application performance monitoring(APM) experience in CloudWatch is delivered through the Amazon Distro for OpenTelemetry (ADOT) SDKs and are used with the CloudWatch Agent to collect application metrics and traces. This option works best if you want to get started with APM in CloudWatch quickly and also leverage out-of-the box integrations with features, such as Container Insights and CloudWatch Logs. For more information, see Enable Application Signals on Amazon EKS Clusters and Enable Application Signals on Amazon EC2, Amazon ECS, or Kubernetes.

Use the OpenTelemetry SDK and Collector

This setup works for the following use cases:

1. You instrumented your application or plan with OpenTelemetry SDKs and currently are using OpenTelemetry Collector.
2. You're using languages, such as Erlang and Rust, that aren't supported by Amazon Distro for OpenTelemetry (ADOT).

For more information, see OpenTelemetry with CloudWatch.

Use the Amazon X-Ray SDK and daemon

This option is best if you instrumented your application using X-Ray SDKs and haven't migrated ADOT SDKs or OpenTelemetry SDKs.

For more information, see Transaction Search.

Feature comparison

Feature	ADOT SDK + CloudWatch Agent	Open Telemetry SDK + OpenTelemetry Collector	X-Ray SDKs
Amazon Support	Yes	Only for data sent to Amazon	Yes
Nonstandard language support	No	Yes	No
Container Insights integration	Yes	No	No
Out of the box logging with CloudWatch Logs	Yes	No	No
Out of the box runtime metrics	Yes	Yes	No
Always gets metrics on 100% of traffic	Yes	Only at 100% sampling rate	Only at 100% sampl rate

Enable Application Signals in your account

If you haven't enabled Application Signals in this account yet, you must grant Application Signals the permissions it needs to discover your services. You need to do this only once for your account.

To enable CloudWatch Application Signals, do the following.

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Services**.
3. Choose **Start discovering your Services**.
4. Select the check box and choose **Start discovering Services**.

Completing this step for the first time in your account creates the **AWSServiceRoleForCloudWatchApplicationSignals** service-linked role. This role grants Application Signals the following permissions:

- `xray:GetServiceGraph`
- `logs:StartQuery`
- `logs:GetQueryResults`
- `cloudwatch:GetMetricData`
- `cloudwatch:ListMetrics`
- `tag:GetResources`

For more information about this role, see Service-linked role permissions for CloudWatch Application Signals.

5. Choose **Enable Application Signals**.

(Optional) Try out Application Signals with a sample app

To try out CloudWatch Application Signals on a sample app before you instrument your own applications with it, follow the instructions in this section. These instructions use scripts to help you create an Amazon EKS cluster, install a sample application, and instrument the sample application to work with Application Signals.

The sample application is a Spring "Pet Clinic" application that is composed of four microservices. These services run on Amazon EKS on Amazon EC2 and leverage Application Signals enablement scripts to enable the cluster with the Java, Python, or .NET auto-instrumentation agent.

Requirements

- Currently, Application Signals monitors only Java, Python, or .NET applications.
- You must have the Amazon CLI installed on the instance. We recommend Amazon CLI version 2, but version 1 should also work. For more information about installing the Amazon CLI, see Install or update the latest version of the Amazon CLI.
- The scripts in this section are intended to be run in Linux and macOS environments. For Windows instances, we recommend that you use an Amazon Cloud9 environment to run these scripts. For more information about Amazon Cloud9, see What is Amazon Cloud9?.

- Install a supported version of `kubectl`. You must use a version of `kubectl` within one minor version difference of your Amazon EKS cluster control plane. For example, a 1.26 `kubectl` client works with Kubernetes 1.25, 1.26, and 1.27 clusters. If you already have an Amazon EKS cluster, you might need to configure Amazon credentials for `kubectl`. For more information, see [Creating or updating a `kubeconfig` file for an Amazon EKS cluster].
- Install `eksctl`. `eksctl` uses the Amazon CLI to interact with Amazon, which means it uses the same Amazon credentials as the Amazon CLI. For more information, see [Installing or updating `eksctl`].
- Install `jq`. `jq` is required to run the Application Signals enablement scripts. For more information, see [Download jq].

Step 1: Download the scripts

To download the scripts to set up CloudWatch Application Signals with a sample app, you can download and uncompress the zipped GitHub project file to a local drive, or you can clone the GitHub project.

To clone the project, open a terminal window and enter the following Git command in a given working directory.

```
git clone https://github.com/aws-observability/application-signals-demo.git
```

Step 2: Build and deploy the sample application

To build and push the sample application images, [follow these instructions].

Step 3: Deploy and enable Application Signals and the sample application

Be sure that you have completed the requirements listed in [(Optional) Try out Application Signals with a sample app] before you complete the following steps.

To deploy and enable Application Signals and the sample application

1. Enter the following command. Replace *`new-cluster-name`* with the name that you want to use for the new cluster. Replace *`region-name`* with the name of the Amazon Region, such as `us-west-1`.

   This command sets up the sample app running in a new Amazon EKS cluster with Application Signals enabled.

```
# this script sets up a new cluster, enables Application Signals, and deploys the
# sample application
cd application-signals-demo/scripts/eks/appsignals/one-step && ./setup.sh new-cluster-name region-name
```

The setup script takes about 30 minutes to run, and does the following:

- Creates a new Amazon EKS cluster in the specified Region.
- Creates the necessary IAM permissions for Application Signals (`arn:aws:iam::aws:policy/AWSXrayWriteOnlyAccess` and `arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy`).
- Enables Application Signals by installing the CloudWatch agent and Auto-instrumenting the sample application for CloudWatch metrics and X-Ray traces.
- Deploys the PetClinic Spring sample application in the same Amazon EKS cluster.
- Creates five CloudWatch Synthetics canaries, named `pc-add-vist`, `pc-create-owners`, `pc-visit-pet`, `pc-visit-vet`, `pc-clinic-traffic`. These canaries will run at a one-minute frequency to generate synthetic traffic for the sample app and demonstrate how Synthetics canaries appear in Application Signals.
- Creates four service level objectives (SLOs) for the PetClinic application with the following names:
  - **Availability for Searching an Owner**
  - **Latency for Searching an Owner**
  - **Availability for Registering an Owner**
  - **Latency for Registering an Owner**
- Creates the required IAM role with a custom trust policy granting Application Signals the following permissions:
  - `cloudwatch:PutMetricData`
  - `cloudwatch:GetMetricData`
  - `xray:GetServiceGraph`
  - `logs:StartQuery`
  - `logs:GetQueryResults`

2. (Optional) If you want to review the source code for the PetClinic sample application, you can find them under the root folder.

```
- application-signals-demo
  - spring-petclinic-admin-server
  - spring-petclinic-api-gateway
  - spring-petclinic-config-server
  - spring-petclinic-customers-service
  - spring-petclinic-discovery-server
  - spring-petclinic-vets-service
  - spring-petclinic-visits-service
```

3. To view the deployed PetClinic sample application, run the following command to find the URL:

```
kubectl get ingress
```

Step 4: Monitor the sample application

After completing the steps in the previous section to create the Amazon EKS cluster and deploy the sample application, you can use Application Signals to monitor the application.

> **Note**
>
> For the Application Signals console to start populating, some traffic must reach the sample application. Part of the previous steps created CloudWatch Synthetics canaries that generate traffic to the sample application.

Service health monitoring

After it is enabled, CloudWatch Application Signals automatically discovers and populates a list of services without requiring any additional setup.

To view the list of discovered services and monitor their health

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Application Signals**, **Services**.
3. To view a service, its operations, and its dependencies, choose the name of one of the services in the list.

This unified, application-centric view helps provide a full perspective of how users are interacting with your service. This can help you triage issues if performance anomalies occur. For complete details about the **Services** view, see [Monitor the operational health of your applications with Application Signals].

4. Choose the **Service Operations** tab to see the standard application metrics for that service's operations. The operations are the API operations that the service calls, for example.

   Then, to view the graphs for a single operation of that service, choose that operation name.

5. Choose the **Dependencies** tab to see the dependencies that your application has, along with the critical application metrics for each dependency. Dependencies include Amazon services and third-party services that your application calls.

6. To view correlated traces from the service details page, choose a data point in one of the three graphs above the table. This populates a new pane with filtered traces from the time period. These traces are sorted and filtered based on the graph that you chose. For example, if you chose the **Latency** graph, the traces are sorted by service response time.

7. In the CloudWatch console navigation pane, choose **SLOs**. You see the SLOs that the script created for the sample application. For more information about SLOs, see [Service level objectives (SLOs)].

(Optional) Step 5: Cleanup

When you're finished testing Application signals, you can use a script provided by Amazon to clean up and delete the artifacts created in your account for the sample application. To perform the cleanup, enter the following command. Replace *`new-cluster-name`* with the name of the cluster that you created for the sample app, and replace *`region`*-name with the name of the Amazon Region, such as `us-west-1`.

```
cd application-signals-demo/scripts/eks/appsignals/one-step && ./cleanup.sh new-cluster-name region-name
```

Enable your applications on Amazon EKS clusters

CloudWatch Application Signals is supported for Java, Python, Node.js, and .NET applications. To enable Application Signals for your applications on an existing Amazon EKS cluster, you can use the Amazon Web Services Management Console or the Amazon CDK.

Topics

Enable Application Signals on an Amazon EKS cluster using the console

To enable CloudWatch Application Signals on your applications on an existing Amazon EKS cluster, use the instructions in this section.

> **Important**
>
> If you are already using OpenTelemetry with an application that you intend to enable for Application Signals, see OpenTelemetry compatibility before you enable Application Signals.

To enable Application Signals for your applications on an existing Amazon EKS cluster

> **Note**
>
> If you haven't already enabled Application Signals, follow the instructions in Enable Application Signals in your account and then follow the procedure below.

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. Choose **Application Signals**.
3. For **Specify platform**, choose **EKS**.
4. For **Select an EKS cluster**, select the cluster where you want to enable Application Signals.
5. If this cluster does not already have the Amazon CloudWatch Observability EKS add-on enabled, you are prompted to enable it. If this is the case, do the following:

    a. Choose **Add CloudWatch Observability EKS add-on**. The Amazon EKS console appears.

    b. Select the check box for **Amazon CloudWatch Observability** and choose **Next**.

       The CloudWatch Observability EKS add-on enables both Application Signals and CloudWatch Container Insights with enhanced observability for Amazon EKS. For more information about Container Insights, see Container Insights.

c. Select the most recent version of the add-on to install.

d. Select an IAM role to use for the add-on. If you choose **Inherit from node**, attach the correct permissions to the IAM role used by your worker nodes. Replace *my-worker-node-role* with the IAM role used by your Kubernetes worker nodes.

```
aws iam attach-role-policy \
--role-name my-worker-node-role \
--policy-arn arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy \
--policy-arn arn:aws:iam::aws:policy/AWSXRayWriteOnlyAccess
```

e. If you want to create a service role to use the add-on, see Install the CloudWatch agent with the Amazon CloudWatch Observability EKS add-on or the Helm chart.

f. Choose **Next**, confirm the information on the screen, and choose **Create**.

g. In the next screen, choose **Enable CloudWatch Application Signals** to return to the CloudWatch console and finish the process.

6. There are two options for enabling your applications for Application Signals. For consistency, we recommend that you choose one option per cluster.

- The **Console** option is simpler. Using this method causes your pods to restart immediately.
- The **Annotate Manifest File** method gives you more control of when your pods restart, and can also help you manage your monitoring in a more decentralized way if you don't want to centralize it.

> **Note**
>
> If you are enabling Application Signals for a Node.js application with ESM, skip to Setting up a Node.js application with the ESM module format instead.

Console

The **Console** option uses the advanced configuration of the Amazon CloudWatch Observability EKS add-on to setup Application Signals for your services. For more information about the add-on, see (Optional) Additional configuration.

If you don't see a list of workloads and namespaces, ensure you have the right permissions to view them for this cluster. For more information, see Required permissions.

You can monitor single workloads or entire namespaces.

To monitor a single workload:

1. Select the check box by the workload that you want to monitor.
2. Use the **Select language(s)** dropdown list to select the language of the workload. Select the languages that you want to enable Application Signals for, and then choose the check mark icon (✓) to save this selection.

   For Python applications, ensure your application follows the required prerequisites before continuing. For more information, see [Python application doesn't start after Application Signals is enabled].
3. Choose **Done**. The Amazon CloudWatch Observability EKS add-on will immediately inject Amazon Distro for OpenTelemetry autoinstrumentation (ADOT) SDKs into your pods and trigger pod restarts to enable collection of application metrics and traces.

To monitor an entire namespace:

1. Select the check box by the namespace that you want to monitor.
2. Use the **Select language(s)** dropdown list to select the language of the namespace. Select the languages that you want to enable Application Signals for, and then choose the check mark icon (✓) to save this selection. This applies it to all workloads in this namespace, whether they are currently deployed or will be deployed in the future.

   For Python applications, ensure your application follows the required prerequisites before continuing. For more information, see [Python application doesn't start after Application Signals is enabled].
3. Choose **Done**. The Amazon CloudWatch Observability EKS add-on will immediately inject Amazon Distro for OpenTelemetry autoinstrumentation (ADOT) SDKs into your pods and trigger pod restarts to enable collection of application metrics and traces.

To enable Application Signals in another Amazon EKS cluster, choose **Enable Application Signals** from the **Services** screen.

Annotate manifest file

In the CloudWatch console, the **Monitor Services** section explains that you must add an annotation to a manifest YAML in the cluster. Adding this annotation auto-instruments the application to send metrics, traces, and logs to Application Signals.

You have two options for the annotation:

- **Annotate Workload** auto-instruments a single workload in the cluster.
- **Annotate Namespace** auto-instruments all workloads deployed in the selected namespace.

Choose one of those options, and follow the appropriate steps:

- To annotate a single workload:
  1. Choose **Annotate Workload**.
  2. Paste one of the following lines into the `PodTemplate` section of the workload manifest file.
     - **For Java workloads:** `annotations: instrumentation.opentelemetry.io/inject-java: "true"`
     - **For Python workloads:** `annotations: instrumentation.opentelemetry.io/inject-python: "true"`

       For Python applications, there are additional required configurations. For more information, see Python application doesn't start after Application Signals is enabled.
     - **For .NET workloads** `annotations: instrumentation.opentelemetry.io/inject-dotnet: "true"`

     > **Note**
     >
     > To enable Application Signals for a .NET workload on Alpine Linux (`linux-musl-x64`) based images, add the following annotation.
     >
     > ```
     > instrumentation.opentelemetry.io/otel-dotnet-auto-runtime: "linux-musl-x64"
     > ```

    - **For Node.js workloads:** `annotations: instrumentation.opentelemetry.io/inject-nodejs: "true"`
  3. In your terminal, enter `kubectl apply -f your_deployment_yaml` to apply the change.
- To annotate all workloads in a namespace:
  1. Choose **Annotate Namespace**.
  2. Paste one of the following lines into the metadata section of the namespace manifest file. If the namespace includes Java, Python, and .NET workloads, paste all of the following lines into the namespace manifest file.
     - **If there are Java workloads in the namespace:** `annotations: instrumentation.opentelemetry.io/inject-java: "true"`
     - **If there are Python workloads in the namespace:** `annotations: instrumentation.opentelemetry.io/inject-python: "true"`

       For Python applications, there are additional required configurations. For more information, see Python application doesn't start after Application Signals is enabled.
     - **If there are .NET workloads in the namespace:** `annotations: instrumentation.opentelemetry.io/inject-dotnet: "true"`
     - **If there are Node.JS workloads in the namespace:** `annotations: instrumentation.opentelemetry.io/inject-nodejs: "true"`
  3. In your terminal, enter `kubectl apply -f your_namespace_yaml` to apply the change.
  4. In your terminal, enter a command to restart all pods in the namespace. An example command to restart deployment workloads is `kubectl rollout restart deployment -n namespace_name`

7. Choose **View Services when done**. This takes you to the Application Signals Services view, where you can see the data that Application Signals is collecting. It might take a few minutes for data to appear.

   To enable Application Signals in another Amazon EKS cluster, choose **Enable Application Signals** from the **Services** screen.

   For more information about the **Services** view, see Monitor the operational health of your applications with Application Signals.

Note

If you're using a WSGI server for your Python application, see No Application Signals data for Python application that uses a WSGI server for information to make Application Signals work.

We've also identified other considerations that you should keep in mind when enabling Python applications for Application Signals. For more information, see Python application doesn't start after Application Signals is enabled.

Setting up a Node.js application with the ESM module format

We provide limited support for Node.js applications with the ESM module format. For details, see the section called "Known limitations about Node.js with ESM".

For the ESM module format, enabling Application Signals through the console or by annotating the manifest file doesn't work. Skip step 8 of the previous procedure, and do the following instead.

To enable Application Signals for a Node.js application with ESM

1. Install the relevant dependencies to your Node.js application for autoinstrumentation:

```
npm install @aws/aws-distro-opentelemetry-node-autoinstrumentation
npm install @opentelemetry/instrumentation@0.54.0
```

2. Add the following environmental variables to the Dockerfile for your application and build the image.

```
...
ENV OTEL_AWS_APPLICATION_SIGNALS_ENABLED=true
ENV OTEL_TRACES_SAMPLER_ARG='endpoint=http://cloudwatch-agent.amazon-
cloudwatch:2000'
ENV OTEL_TRACES_SAMPLER='xray'
ENV OTEL_EXPORTER_OTLP_PROTOCOL='http/protobuf'
ENV OTEL_EXPORTER_OTLP_TRACES_ENDPOINT='http://cloudwatch-agent.amazon-
cloudwatch:4316/v1/traces'
ENV OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT='http://cloudwatch-agent.amazon-
cloudwatch:4316/v1/metrics'
ENV OTEL_METRICS_EXPORTER='none'
ENV OTEL_LOGS_EXPORTER='none'
```

```
ENV NODE_OPTIONS='--import @aws/aws-distro-opentelemetry-node-autoinstrumentation/
register --experimental-loader=@opentelemetry/instrumentation/hook.mjs'
ENV OTEL_SERVICE_NAME='YOUR_SERVICE_NAME' #replace with a proper service name
ENV OTEL_PROPAGATORS='tracecontext,baggage,b3,xray'
...

# command to start the application
# for example
# CMD ["node", "index.mjs"]
```

3. Add the environmental variables `OTEL_RESOURCE_ATTRIBUTES_POD_NAME`, `OTEL_RESOURCE_ATTRIBUTES_NODE_NAME`, `OTEL_RESOURCE_ATTRIBUTES_DEPLOYMENT_NAME`, `POD_NAMESPACE` and `OTEL_RESOURCE_ATTRIBUTES` to the deployment yaml file for the application. For example:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nodejs-app
  labels:
    app: nodejs-app
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nodejs-app
  template:
    metadata:
      labels:
        app: nodejs-app
      # annotations:
      # make sure this annotation doesn't exit
      #   instrumentation.opentelemetry.io/inject-nodejs: 'true'
    spec:
      containers:
      - name: nodejs-app
        image:your-nodejs-application-image #replace with a proper image uri
        imagePullPolicy: Always
        ports:
        - containerPort: 8000
        env:
          - name: OTEL_RESOURCE_ATTRIBUTES_POD_NAME
            valueFrom:
```

```
              fieldRef:
                fieldPath: metadata.name
          - name: OTEL_RESOURCE_ATTRIBUTES_NODE_NAME
            valueFrom:
              fieldRef:
                fieldPath: spec.nodeName
          - name: OTEL_RESOURCE_ATTRIBUTES_DEPLOYMENT_NAME
            valueFrom:
              fieldRef:
                fieldPath: metadata.labels['app'] # Assuming 'app' label is set to
 the deployment name
          - name: POD_NAMESPACE
            valueFrom:
              fieldRef:
                fieldPath: metadata.namespace
          - name: OTEL_RESOURCE_ATTRIBUTES
            value: "k8s.deployment.name=
$(OTEL_RESOURCE_ATTRIBUTES_DEPLOYMENT_NAME),k8s.namespace.name=
$(POD_NAMESPACE),k8s.node.name=$(OTEL_RESOURCE_ATTRIBUTES_NODE_NAME),k8s.pod.name=
$(OTEL_RESOURCE_ATTRIBUTES_POD_NAME)"
```

4. Deploy the Node.js application to the cluster.

Once you have enabled your applications on the Amazon EKS Clusters, you can monitor your application health. For more information, see Monitor the operational health of your applications with Application Signals.

Enable Application Signals on Amazon EKS using Amazon CDK

If you haven't enabled Application Signals in this account yet, you must grant Application Signals the permissions it needs to discover your services. See Enable Application Signals in your account.

1. Enable Application Signals for your applications.

```
import { aws_applicationsignals as applicationsignals } from 'aws-cdk-lib';

const cfnDiscovery = new applicationsignals.CfnDiscovery(this,
  'ApplicationSignalsServiceRole', { }
);
```

The Discovery CloudFormation resource grants Application Signals the following permissions:

- `xray:GetServiceGraph`
- `logs:StartQuery`
- `logs:GetQueryResults`
- `cloudwatch:GetMetricData`
- `cloudwatch:ListMetrics`
- `tag:GetResources`

For more information about this role, see Service-linked role permissions for CloudWatch Application Signals.

2. Install the `amazon-cloudwatch-observability` add-on.

   - Create an IAM role with the `CloudWatchAgentServerPolicy` and the OIDC associated with the cluster.

```
const cloudwatchRole = new Role(this, 'CloudWatchAgentAddOnRole', {
    assumedBy: new OpenIdConnectPrincipal(cluster.openIdConnectProvider),
    managedPolicies:
 [ManagedPolicy.fromAwsManagedPolicyName('CloudWatchAgentServerPolicy')],
});
```

3. Install the add-on with the IAM role created above.

```
new CfnAddon(this, 'CloudWatchAddon', {
    addonName: 'amazon-cloudwatch-observability',
    clusterName: cluster.clusterName,
    serviceAccountRoleArn: cloudwatchRole.roleArn
});
```

4. Add one of the following into the `PodTemplate` section of your workload manifest file.

Language	File
Java	instrumentation.opentelemetry.io/inject-java: "true"
Python	instrumentation.opentelemetry.io/inject-python: "true"

Language	File
.Net	instrumentation.opentelemetry.io/inject-dotnet: "true"
Node.js	instrumentation.opentelemetry.io/inject-nodejs: "true"

```
const deployment = {
  apiVersion: "apps/v1",
  kind: "Deployment",
  metadata: { name: "sample-app" },
  spec: {
    replicas: 3,
    selector: {
      matchLabels: {
        "app": "sample-app"
      }
    },
    template: {
      metadata: {
        labels: {
          "app": "sample-app"
        },
        annotations: {
          "instrumentation.opentelemetry.io/inject-$LANG": "true"
        }
      },
      spec: {...},
    },
  },
};

cluster.addManifest('sample-app', deployment)
```

5.

Enable your applications on Amazon EC2

Enable CloudWatch Application Signals on Amazon EC2 by using the custom setup steps described in this section.

For applications running on Amazon EC2, you install and configure the CloudWatch agent and Amazon Distro for OpenTelemetry yourself. On these architectures enabled with a custom Application Signals setup, Application Signals doesn't autodiscover the names of your services or the hosts or clusters they run on. You must specify these names during the custom setup, and the names that you specify are what is displayed on Application Signals dashboards.

The instructions in this section are for Java, Python, and .NET applications. The steps have been tested on Amazon EC2 instances, but are also expected to work on other architectures that support Amazon Distro for OpenTelemetry.

Requirements

- To get support for Application Signals, you must use the most recent version of both the CloudWatch agent and the Amazon Distro for OpenTelemetry agent.
- You must have the Amazon CLI installed on the instance. We recommend Amazon CLI version 2, but version 1 should also work. For more information about installing the Amazon CLI, see Install or update the latest version of the Amazon CLI.

> **Important**
>
> If you are already using OpenTelemetry with an application that you intend to enable for Application Signals, see OpenTelemetry compatibility before you enable Application Signals.

Step 1: Enable Application Signals in your account

You must first enable Application Signals in your account. If you haven't, see Enable Application Signals in your account.

Step 2: Download and start the CloudWatch agent

To install the CloudWatch agent as part of enabling Application Signals on an Amazon EC2 instance or on-premises host

1. Download the latest version of the CloudWatch agent to the instance. If the instance already has the CloudWatch agent installed, you might need to update it. Only versions of the agent released on November 30, 2023 or later support CloudWatch Application Signals.

   For information about downloading the CloudWatch agent, see Download the CloudWatch agent package.

2. Before you start the CloudWatch agent, configure it to enable Application Signals. The following example is a CloudWatch agent configuration that enables Application Signals for both metrics and traces on an EC2 host.

   We recommend that you place this file at `/opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.json` on Linux systems.

```
{
  "traces": {
    "traces_collected": {
      "application_signals": {}
    }
  },
  "logs": {
    "metrics_collected": {
      "application_signals": {}
    }
  }
}
```

3. Attach the **CloudWatchAgentServerPolicy** IAM policy to the IAM role of your Amazon EC2 instance. For permissions for on-premises hosts, see Permissions for on-premises servers.

   a. Sign in to the Amazon Web Services Management Console and open the IAM console at https://console.amazonaws.cn/iam/.

   b. Choose **Roles** and find the role used by your Amazon EC2 instance. Then choose the name of that role.

   c. In the **Permissions** tab, choose **Add permissions**, **Attach policies**.

   d. Find **CloudWatchAgentServerPolicy**. Use the search box if needed. Then select the check box for that policy and choose **Add permissions**.

4. Start the CloudWatch agent by entering the following commands. Replace *agent-config-file-path* with the path to the CloudWatch agent configuration file, such as `./amazon-cloudwatch-agent.json`. You must include the `file:` prefix as shown.

```
export CONFIG_FILE_PATH=./amazon-cloudwatch-agent.json
```

```
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl \
-a fetch-config \
-m ec2 -s -c file:agent-config-file-path
```

Permissions for on-premises servers

For an on-premises host, you will need to provide Amazon authorization to your device.

To set up permissions for an on-premises host

1. Create the IAM user to be used to provide permissions to your on-premises host:

   a. Open the IAM console at https://console.amazonaws.cn/iam/.
   b. Choose **Users**, **Create User**.
   c. In **User details**, for **User name**, enter a name for the new IAM user. This is the sign-in name for Amazon that will be used to authenticate your host. Then choose **Next**
   d. On the **Set permissions** page, under **Permissions options**, select **Attach policies directly**.
   e. From the **Permissions policies** list, select the **CloudWatchAgentServerPolicy** policy to add to your user. Then choose **Next**.
   f. On the **Review and create** page, ensure that you are satisfied with the user name and that the **CloudWatchAgentServerPolicy** policy is in the **Permissions summary**.
   g. Choose **Create user**

2. Create and retrieve your Amazon access key and secret key:

   a. In the navigation pane in the IAM console, choose **Users** and then select the user name of the user that you created in the previous step.
   b. On the user's page, choose the **Security credentials** tab. Then, in the **Access keys** section, choose **Create access key**.

c. For **Create access key Step 1**, choose **Command Line Interface (CLI)**.

d. For **Create access key Step 2**, optionally enter a tag and then choose **Next**.

e. For **Create access key Step 3**, select **Download .csv file** to save a .csv file with your IAM user's access key and secret access key. You need this information for the next steps.

f. Choose **Done**.

3. Configure your Amazon credentials in your on-premises host by entering the following command. Replace *ACCESS_KEY_ID* and *SECRET_ACCESS_ID* with your newly generated access key and secret access key from the .csv file that you downloaded in the previous step.

```
$ aws configure
AWS Access Key ID [None]: ACCESS_KEY_ID
AWS Secret Access Key [None]: SECRET_ACCESS_ID
Default region name [None]: MY_REGION
Default output format [None]: json
```

Step 3: Instrument your application and start it

The next step is to instrument your application for CloudWatch Application Signals.

Java

To instrument your Java applications as part of enabling Application Signals on an Amazon EC2 instance or on-premises host

1. Download the latest version of the Amazon Distro for OpenTelemetry Java auto-instrumentation agent. You can download the latest version by using this link. You can view information about all released versions at aws-otel-java-instrumentation Releases.

2. To optimize your Application Signals benefits, use environment variables to provide additional information before you start your application. This information will be displayed in Application Signals dashboards.

   - For the `OTEL_RESOURCE_ATTRIBUTES` variable, specify the following information as key-value pairs:

     - (Optional) `service.name` sets the name of the service. This will be displayed as the service name for your application in Application Signals dashboards. If you don't provide a value for this key, the default of `UnknownService` is used.

- (Optional) `deployment.environment` sets the environment that the application runs in. This will be diplayed as the **Hosted In** environment of your application in Application Signals dashboards. If you don't specify this, one of the following defaults is used:
  - If this is an instance that is part of an Auto Scaling group, it is set to `ec2:`*`name-of-Auto-Scaling-group`*
  - If this is an Amazon EC2 instance that is not part of an Auto Scaling group, it is set to `ec2:default`
  - If this is an on-premises host, it is set to `generic:default`

  This environment variable is used only by Application Signals, and is converted into X-Ray trace annotations and CloudWatch metric dimensions.
- For the `OTEL_EXPORTER_OTLP_TRACES_ENDPOINT` variable, specify the base endpoint URL where traces are to be exported to. The CloudWatch agent exposes 4316 as its OTLP port. On Amazon EC2, because applications communicate with the local CloudWatch agent, you should set this value to `OTEL_EXPORTER_OTLP_TRACES_ENDPOINT=http://localhost:4316/v1/traces`
- For the `OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT` variable, specify the base endpoint URL where metrics are to be exported to. The CloudWatch agent exposes 4316 as its OTLP port. On Amazon EC2, because applications communicate with the local CloudWatch agent, you should set this value to `OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT=http://localhost:4316/v1/metrics`
- For the `JAVA_TOOL_OPTIONS` variable, specify the path where the Amazon Distro for OpenTelemetry Java auto-instrumentation agent is stored.

  ```
  export JAVA_TOOL_OPTIONS=" -javaagent:$AWS_ADOT_JAVA_INSTRUMENTATION_PATH"
  ```

  For example:

  ```
  export AWS_ADOT_JAVA_INSTRUMENTATION_PATH=./aws-opentelemetry-agent.jar
  ```
- For the `OTEL_METRICS_EXPORTER` variable, we recommend that you set the value to `none`. This disables other metrics exporters so that only the Application Signals exporter is used.

- Set `OTEL_AWS_APPLICATION_SIGNALS_ENABLED` to `true`. This generates Application Signals metrics from traces.

3. Start your application with the environment variables listed in the previous step. The following is an example of a starting script.

> **Note**
>
> The following configuration supports only versions 1.32.2 and later of the Amazon Distro for OpenTelemetry auto-instrumentation agent for Java.

```
JAVA_TOOL_OPTIONS=" -javaagent:$AWS_ADOT_JAVA_INSTRUMENTATION_PATH" \
OTEL_METRICS_EXPORTER=none \
OTEL_LOGS_EXPORTER=none \
OTEL_AWS_APPLICATION_SIGNALS_ENABLED=true \
OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT=http://localhost:4316/v1/metrics
 \
OTEL_EXPORTER_OTLP_PROTOCOL=http/protobuf \
OTEL_EXPORTER_OTLP_TRACES_ENDPOINT=http://localhost:4316/v1/traces \
OTEL_RESOURCE_ATTRIBUTES="service.name=$YOUR_SVC_NAME" \
java -jar $MY_JAVA_APP.jar
```

4. (Optional) To enable log correlation, in `OTEL_RESOURCE_ATTRIBUTES`, set an additional environment variable `aws.log.group.names` for the log groups of your application. By doing so, the traces and metrics from your application can be correlated with the relevant log entries from these log groups. For this variable, replace *`$YOUR_APPLICATION_LOG_GROUP`* with the log group names for your application. If you have multiple log groups, you can use an ampersand (&) to separate them as in this example: `aws.log.group.names=log-group-1&log-group-2`. To enable metric to log correlation, setting this current environmental variable is enough. For more information, see Enable metric to log correlation. To enable trace to log correlation, you'll also need to change the logging configuration in your application. For more information, see Enable trace to log correlation.

   The following is an example of a starting script that helps enable log correlation.

```
JAVA_TOOL_OPTIONS=" -javaagent:$AWS_ADOT_JAVA_INSTRUMENTATION_PATH" \
OTEL_METRICS_EXPORTER=none \
OTEL_LOGS_EXPORT=none \
```

```
OTEL_AWS_APPLICATION_SIGNALS_ENABLED=true \
OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT=http://localhost:4316/v1/metrics
 \
OTEL_EXPORTER_OTLP_PROTOCOL=http/protobuf \
OTEL_EXPORTER_OTLP_TRACES_ENDPOINT=http://localhost:4316/v1/traces \
OTEL_RESOURCE_ATTRIBUTES="aws.log.group.names=$YOUR_APPLICATION_LOG_GROUP,service.name=$
 \
java -jar $MY_JAVA_APP.jar
```

Python

Note

If you're using a WSGI server for your Python application, in addition to the following steps in this section, see No Application Signals data for Python application that uses a WSGI server for information to make Application Signals work.

To instrument your Python applications as part of enabling Application Signals on an Amazon EC2 instance

1. Download the latest version of the Amazon Distro for OpenTelemetry Python auto-instrumentation agent. Install it by running the following command.

   ```
   pip install aws-opentelemetry-distro
   ```

   You can view information about all released versions at Amazon Distro for OpenTelemetry Python instrumentation.

2. To optimize your Application Signals benefits, use environment variables to provide additional information before you start your application. This information will be displayed in Application Signals dashboards.

   a. For the `OTEL_RESOURCE_ATTRIBUTES` variable, specify the following information as key-value pairs:

      - `service.name` sets the name of the service. This will be diplayed as the service name for your application in Application Signals dashboards. If you don't provide a value for this key, the default of `UnknownService` is used.

- `deployment.environment` sets the environment that the application runs in. This will be diplayed as the **Hosted In** environment of your application in Application Signals dashboards. If you don't specify this, one of the following defaults is used:
  - If this is an instance that is part of an Auto Scaling group, it is set to `ec2:`*`name-of-Auto-Scaling-group`*.
  - If this is an Amazon EC2 instance that is not part of an Auto Scaling group, it is set to `ec2:default`
  - If this is an on-premises host, it is set to `generic:default`

  This attribute key is used only by Application Signals, and is converted into X-Ray trace annotations and CloudWatch metric dimensions.

b. For the `OTEL_EXPORTER_OTLP_PROTOCOL` variable, specify `http/protobuf` to export telemetry data over HTTP to the CloudWatch agent endpoints listed in the following steps.

c. For the `OTEL_EXPORTER_OTLP_TRACES_ENDPOINT` variable, specify the base endpoint URL where traces are to be exported to. The CloudWatch agent exposes 4316 as its OTLP port over HTTP. On Amazon EC2, because applications communicate with the local CloudWatch agent, you should set this value to `OTEL_EXPORTER_OTLP_TRACES_ENDPOINT=http://localhost:4316/v1/traces`

d. For the `OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT` variable, specify the base endpoint URL where metrics are to be exported to. The CloudWatch agent exposes 4316 as its OTLP port over HTTP. On Amazon EC2, because applications communicate with the local CloudWatch agent, you should set this value to `OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT=http://localhost:4316/v1/metrics`

e. For the `OTEL_METRICS_EXPORTER` variable, we recommend that you set the value to none. This disables other metrics exporters so that only the Application Signals exporter is used.

f. Set the `OTEL_AWS_APPLICATION_SIGNALS_ENABLED` variable to `true` to have your container start sending X-Ray traces and CloudWatch metrics to Application Signals.

3. Start your application with the environment variables discussed in the previous step. The following is an example of a starting script.

- Replace $SVC_NAME with your application name. This will be displayed as the name of the application, in Application Signals dashboards.
- Replace $PYTHON_APP with the location and name of your application.

```
OTEL_METRICS_EXPORTER=none \
OTEL_LOGS_EXPORTER=none \
OTEL_AWS_APPLICATION_SIGNALS_ENABLED=true \
OTEL_PYTHON_DISTRO=aws_distro \
OTEL_PYTHON_CONFIGURATOR=aws_configurator \
OTEL_EXPORTER_OTLP_PROTOCOL=http/protobuf \
OTEL_TRACES_SAMPLER=xray \
OTEL_TRACES_SAMPLER_ARG="endpoint=http://localhost:2000" \
OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT=http://localhost:4316/v1/metrics
 \
OTEL_EXPORTER_OTLP_TRACES_ENDPOINT=http://localhost:4316/v1/traces \
OTEL_RESOURCE_ATTRIBUTES="service.name=$SVC_NAME" \
opentelemetry-instrument python $MY_PYTHON_APP.py
```

Before you enable Application Signals for your Python applications, be aware of the following considerations.

- In some containerized applications, a missing PYTHONPATH environment variable can sometimes cause the application to fail to start. To resolve this, ensure that you set the PYTHONPATH environment variable to the location of your application's working directory. This is due to a known issue with OpenTelemetry auto-instrumentation. For more information about this issue, see Python autoinstrumentation setting of PYTHONPATH is not compliant.
- For Django applications, there are additional required configurations, which are outlined in the OpenTelemetry Python documentation.
  - Use the `--noreload` flag to prevent automatic reloading.
  - Set the DJANGO_SETTINGS_MODULE environment variable to the location of your Django application's `settings.py` file. This ensures that OpenTelemetry can correctly access and integrate with your Django settings.

4. (Optional) To enable log correlation, in OTEL_RESOURCE_ATTRIBUTES, set an additional environment variable `aws.log.group.names` for the log groups of your application. By doing so, the traces and metrics from your application can be

correlated with the relevant log entries from these log groups. For this variable, replace *$YOUR_APPLICATION_LOG_GROUP* with the log group names for your application. If you have multiple log groups, you can use an ampersand (&) to separate them as in this example: `aws.log.group.names=log-group-1&log-group-2`. To enable metric to log correlation, setting this current environmental variable is enough. For more information, see Enable metric to log correlation. To enable trace to log correlation, you'll also need to change the logging configuration in your application. For more information, see Enable trace to log correlation.

The following is an example of a starting script that helps enable log correlation.

```
OTEL_METRICS_EXPORTER=none \
OTEL_LOGS_EXPORTER=none \
OTEL_AWS_APPLICATION_SIGNALS_ENABLED=true \
OTEL_PYTHON_DISTRO=aws_distro \
OTEL_PYTHON_CONFIGURATOR=aws_configurator \
OTEL_EXPORTER_OTLP_PROTOCOL=http/protobuf \
OTEL_TRACES_SAMPLER=xray \
OTEL_TRACES_SAMPLER_ARG="endpoint=http://localhost:2000" \
OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT=http://localhost:4316/v1/metrics
 \
OTEL_EXPORTER_OTLP_TRACES_ENDPOINT=http://localhost:4316/v1/traces \
OTEL_RESOURCE_ATTRIBUTES="aws.log.group.names=$YOUR_APPLICATION_LOG_GROUP,service.name=$
 \
java -jar $MY_PYTHON_APP.jar
```

.NET

To instrument your .NET applications as part of enabling Application Signals on an Amazon EC2 instance or on-premises host

1. Download the latest version of the Amazon Distro for OpenTelemetry .NET auto-instrumentation package. You can download the latest version at aws-otel-dotnet-instrumentation Releases.
2. To enable Application Signals, set the following environment variables to provide additional information before you start your application. These variables are necessary to set up the startup hook for .NET instrumentation, before you start your .NET application. Replace `dotnet-service-name` in the `OTEL_RESOURCE_ATTRIBUTES` environment variable with the service name of your choice.

- The following is an example for Linux.

```
export INSTALL_DIR=OpenTelemetryDistribution
export CORECLR_ENABLE_PROFILING=1
export CORECLR_PROFILER={918728DD-259F-4A6A-AC2B-B85E1B658318}
export CORECLR_PROFILER_PATH=${INSTALL_DIR}/linux-x64/
OpenTelemetry.AutoInstrumentation.Native.so
export DOTNET_ADDITIONAL_DEPS=${INSTALL_DIR}/AdditionalDeps
export DOTNET_SHARED_STORE=${INSTALL_DIR}/store
export DOTNET_STARTUP_HOOKS=${INSTALL_DIR}/net/
OpenTelemetry.AutoInstrumentation.StartupHook.dll
export OTEL_DOTNET_AUTO_HOME=${INSTALL_DIR}

export
 OTEL_DOTNET_AUTO_PLUGINS="AWS.Distro.OpenTelemetry.AutoInstrumentation.Plugin,
 AWS.Distro.OpenTelemetry.AutoInstrumentation"

export OTEL_RESOURCE_ATTRIBUTES=service.name=dotnet-service-name
export OTEL_EXPORTER_OTLP_PROTOCOL=http/protobuf
export OTEL_EXPORTER_OTLP_ENDPOINT=http://127.0.0.1:4316
export OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT=http://127.0.0.1:4316/
v1/metrics
export OTEL_METRICS_EXPORTER=none
export OTEL_AWS_APPLICATION_SIGNALS_ENABLED=true
export OTEL_TRACES_SAMPLER=xray
export OTEL_TRACES_SAMPLER_ARG=http://127.0.0.1:2000
```

- The following is an example for Windows Server.

```
$env:INSTALL_DIR = "OpenTelemetryDistribution"
$env:CORECLR_ENABLE_PROFILING = 1
$env:CORECLR_PROFILER = "{918728DD-259F-4A6A-AC2B-B85E1B658318}"
$env:CORECLR_PROFILER_PATH = Join-Path $env:INSTALL_DIR "win-x64/
OpenTelemetry.AutoInstrumentation.Native.dll"
$env:DOTNET_ADDITIONAL_DEPS = Join-Path $env:INSTALL_DIR "AdditionalDeps"
$env:DOTNET_SHARED_STORE = Join-Path $env:INSTALL_DIR "store"
$env:DOTNET_STARTUP_HOOKS = Join-Path $env:INSTALL_DIR "net/
OpenTelemetry.AutoInstrumentation.StartupHook.dll"
$env:OTEL_DOTNET_AUTO_HOME = $env:INSTALL_DIR
```

```
$env:OTEL_DOTNET_AUTO_PLUGINS =
 "AWS.Distro.OpenTelemetry.AutoInstrumentation.Plugin,
 AWS.Distro.OpenTelemetry.AutoInstrumentation"

$env:OTEL_RESOURCE_ATTRIBUTES = "service.name=dotnet-service-name"
$env:OTEL_EXPORTER_OTLP_PROTOCOL = "http/protobuf"
$env:OTEL_EXPORTER_OTLP_ENDPOINT = "http://127.0.0.1:4316"
$env:OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT = "http://127.0.0.1:4316/
v1/metrics"
$env:OTEL_METRICS_EXPORTER = "none"
$env:OTEL_AWS_APPLICATION_SIGNALS_ENABLED = "true"
$env:OTEL_TRACES_SAMPLER = "xray"
$env:OTEL_TRACES_SAMPLER_ARG = "http://127.0.0.1:2000"
```

3. Start your application with the environment variables listed in the previous step.

   (Optional) Alternatively, you can use the installation scripts provided to help installation and setup of Amazon Distro for OpenTelemetry .NET auto-instrumentation package.

   For Linux, download and install the Bash installation script from the GitHub releases page:

```
# Download and Install
curl -L -O https://github.com/aws-observability/aws-otel-dotnet-instrumentation/
releases/latest/download/aws-otel-dotnet-install.sh
chmod +x ./aws-otel-dotnet-install.sh
./aws-otel-dotnet-install.sh

# Instrument
. $HOME/.otel-dotnet-auto/instrument.sh
export OTEL_RESOURCE_ATTRIBUTES=service.name=dotnet-service-name
```

   For Windows Server, download and install the PowerShell installation script from the GitHub releases page:

```
# Download and Install
$module_url = "https://github.com/aws-observability/aws-otel-dotnet-
instrumentation/releases/latest/download/AWS.Otel.DotNet.Auto.psm1"
$download_path = Join-Path $env:temp "AWS.Otel.DotNet.Auto.psm1"
Invoke-WebRequest -Uri $module_url -OutFile $download_path
Import-Module $download_path
Install-OpenTelemetryCore
```

```
# Instrument
Import-Module $download_path
Register-OpenTelemetryForCurrentSession -OTelServiceName "dotnet-service-name"
Register-OpenTelemetryForIIS
```

You can find the NuGet package of the Amazon Distro for OpenTelemetry .NET auto-instrumentation package in the official NuGet repository. Be sure to check the README file for instructions.

Node.js

Note

If you are enabling Application Signals for a Node.js application with ESM, see Setting up a Node.js application with the ESM module format before you start these steps.

To instrument your Node.js applications as part of enabling Application Signals on an Amazon EC2 instance

1. Download the latest version of the Amazon Distro for OpenTelemetry JavaScript auto-instrumentation agent for Node.js. Install it by running the following command.

   ```
   npm install @aws/aws-distro-opentelemetry-node-autoinstrumentation
   ```

   You can view information about all released versions at Amazon Distro for OpenTelemetry JavaScript instrumentation.

2. To optimize your Application Signals benefits, use environment variables to provide additional information before you start your application. This information will be displayed in Application Signals dashboards.

   a. For the `OTEL_RESOURCE_ATTRIBUTES` variable, specify the following information as key-value pairs:

      - `service.name` sets the name of the service. This will be diplayed as the service name for your application in Application Signals dashboards. If you don't provide a value for this key, the default of `UnknownService` is used.

- `deployment.environment` sets the environment that the application runs in. This will be diplayed as the **Hosted In** environment of your application in Application Signals dashboards. If you don't specify this, one of the following defaults is used:
  - If this is an instance that is part of an Auto Scaling group, it is set to `ec2:name-of-Auto-Scaling-group`.
  - If this is an Amazon EC2 instance that is not part of an Auto Scaling group, it is set to `ec2:default`
  - If this is an on-premises host, it is set to `generic:default`

  This attribute key is used only by Application Signals, and is converted into X-Ray trace annotations and CloudWatch metric dimensions.

b. For the `OTEL_EXPORTER_OTLP_PROTOCOL` variable, specify `http/protobuf` to export telemetry data over HTTP to the CloudWatch agent endpoints listed in the following steps.

c. For the `OTEL_EXPORTER_OTLP_TRACES_ENDPOINT` variable, specify the base endpoint URL where traces are to be exported to. The CloudWatch agent exposes 4316 as its OTLP port over HTTP. On Amazon EC2, because applications communicate with the local CloudWatch agent, you should set this value to `OTEL_EXPORTER_OTLP_TRACES_ENDPOINT=http://localhost:4316/v1/traces`

d. For the `OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT` variable, specify the base endpoint URL where metrics are to be exported to. The CloudWatch agent exposes 4316 as its OTLP port over HTTP. On Amazon EC2, because applications communicate with the local CloudWatch agent, you should set this value to `OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT=http://localhost:4316/v1/metrics`

e. For the `OTEL_METRICS_EXPORTER` variable, we recommend that you set the value to none. This disables other metrics exporters so that only the Application Signals exporter is used.

f. Set the `OTEL_AWS_APPLICATION_SIGNALS_ENABLED` variable to `true` to have your container start sending X-Ray traces and CloudWatch metrics to Application Signals.

3. Start your application with the environment variables discussed in the previous step. The following is an example of a starting script.

- Replace $SVC_NAME with your application name. This will be displayed as the name of the application, in Application Signals dashboards.

```
OTEL_METRICS_EXPORTER=none \
OTEL_LOGS_EXPORTER=none \
OTEL_AWS_APPLICATION_SIGNALS_ENABLED=true \
OTEL_EXPORTER_OTLP_PROTOCOL=http/protobuf \
OTEL_TRACES_SAMPLER=xray \
OTEL_TRACES_SAMPLER_ARG="endpoint=http://localhost:2000" \
OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT=http://localhost:4316/v1/metrics
 \
OTEL_EXPORTER_OTLP_TRACES_ENDPOINT=http://localhost:4316/v1/traces \
OTEL_RESOURCE_ATTRIBUTES="service.name=$SVC_NAME" \
node --require '@aws/aws-distro-opentelemetry-node-autoinstrumentation/
register' your-application.js
```

4. (Optional) To enable log correlation, in `OTEL_RESOURCE_ATTRIBUTES`, set an additional environment variable `aws.log.group.names` for the log groups of your application. By doing so, the traces and metrics from your application can be correlated with the relevant log entries from these log groups. For this variable, replace *$YOUR_APPLICATION_LOG_GROUP* with the log group names for your application. If you have multiple log groups, you can use an ampersand (&) to separate them as in this example: `aws.log.group.names=log-group-1&log-group-2`. To enable metric to log correlation, setting this current environmental variable is enough. For more information, see Enable metric to log correlation. To enable trace to log correlation, you'll also need to change the logging configuration in your application. For more information, see Enable trace to log correlation.

   The following is an example of a starting script that helps enable log correlation.

```
export OTEL_METRICS_EXPORTER=none \
export OTEL_LOGS_EXPORTER=none \
export OTEL_AWS_APPLICATION_SIGNALS_ENABLED=true \
export OTEL_EXPORTER_OTLP_PROTOCOL=http/protobuf \
export OTEL_TRACES_SAMPLER=xray \
export OTEL_TRACES_SAMPLER_ARG=endpoint=http://localhost:2000 \
export OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT=http://localhost:4316/v1/
metrics \
export OTEL_EXPORTER_OTLP_TRACES_ENDPOINT=http://localhost:4316/v1/traces \
```

```
export
 OTEL_RESOURCE_ATTRIBUTES="aws.log.group.names=$YOUR_APPLICATION_LOG_GROUP,service.name=
 \
node --require '@aws/aws-distro-opentelemetry-node-autoinstrumentation/
register' your-application.js
```

Setting up a Node.js application with the ESM module format

We provide limited support for Node.js applications with the ESM module format. For details, see the section called "Known limitations about Node.js with ESM".

To enable Application Signals for a Node.js application with ESM, you need to modify the steps in the previous procedure.

First, install `@opentelemetry/instrumentation` for your Node.js application:

```
npm install @opentelemetry/instrumentation@0.54.0
```

Then, in steps 3 and 4 in the previous procedure, change the node options from:

```
--require '@aws/aws-distro-opentelemetry-node-autoinstrumentation/register'
```

to the following:

```
--import @aws/aws-distro-opentelemetry-node-autoinstrumentation/register --
experimental-loader=@opentelemetry/instrumentation/hook.mjs
```

(Optional) Step 4: Monitor your application health

Once you have enabled your applications on Amazon EC2, you can monitor your application health. For more information, see Monitor the operational health of your applications with Application Signals.

Enable your applications on Amazon ECS

Enable CloudWatch Application Signals on Amazon ECS by using the custom setup steps described in this section.

For applications running on Amazon ECS, you install and configure the CloudWatch agent and Amazon Distro for OpenTelemetry yourself. On these architectures enabled with a custom Application Signals setup, Application Signals doesn't autodiscover the names of your services or the hosts or clusters they run on. You must specify these names during the custom setup, and the names that you specify are what is displayed on Application Signals dashboards.

Use a custom setup to enable Application Signals on Amazon ECS

Use these custom setup instructions to onboard your applications on Amazon ECS to CloudWatch Application Signals. You install and configure the CloudWatch agent and Amazon Distro for OpenTelemetry yourself.

There are two methods for deploying Application Signals on Amazon ECS. Choose the one that is best for your environment.

- Deploy using the sidecar strategy – You add a CloudWatch agent sidecar container to each task definition in the cluster.

  Advantages:

  - Supports both the `ec2` and `Fargate` launch types.
  - You can always use `localhost` as the IP address when you set up environment variables.

  Disadvantages:

  - You must set up the CloudWatch agent sidecar container for each service task that runs in the cluster.
  - Only the `awsvpc` network mode is supported.
- Deploy using the daemon strategy – You add a CloudWatch agent task only once in the cluster, and the Amazon ECS daemon scheduling strategy deploys it as needed. The ensures that each instance continuously receives traces and metrics, providing centralized visibility without the need for the agent to run as a sidecar with each application task definition.

  Advantages:

  - You need to set up the daemon service for the CloudWatch agent only once in the cluster.

  Disadvantages:

  - Doesn't support the Fargate launch type.
  - If you use the `awsvpc` or `bridge` network mode, you have to manually specify each container instance's private IP address in the environment variables.

With either method, on Amazon ECS clusters Application Signals doesn't autodiscover the names of your services. You must specify your service names during the custom setup, and the names that you specify are what is displayed on Application Signals dashboards.

Deploy using the sidecar strategy

Step 1: Enable Application Signals in your account

You must first enable Application Signals in your account. If you haven't, see Enable Application Signals in your account.

Step 2: Create IAM roles

You must create an IAM role. If you already have created this role, you might need to add permissions to it.

- **ECS task role—** Containers use this role to run. The permissions should be whatever your applications need, plus **CloudWatchAgentServerPolicy**.

For more information about creating IAM roles, see Creating IAM Roles.

Step 3: Prepare CloudWatch agent configuration

First, prepare the agent configuration with Application Signals enabled. To do this, create a local file named `/tmp/ecs-cwagent.json`.

```
{
  "traces": {
    "traces_collected": {
      "application_signals": {}
    }
  },
  "logs": {
    "metrics_collected": {
      "application_signals": {}
    }
  }
}
```

Then upload this configuration to the SSM Parameter Store. To do this, enter the following command. In the file, replace *$REGION* with your actual Region name.

```
aws ssm put-parameter \
--name "ecs-cwagent" \
--type "String" \
--value "`cat /tmp/ecs-cwagent.json`" \
--region "$REGION"
```

Step 4: Instrument your application with the CloudWatch agent

The next step is to instrument your application for CloudWatch Application Signals.

Java

To instrument your application on Amazon ECS with the CloudWatch agent

1. First, specify a bind mount. The volume will be used to share files across containers in the next steps. You will use this bind mount later in this procedure.

   ```
   "volumes": [
     {
       "name": "opentelemetry-auto-instrumentation"
     }
   ]
   ```

2. Add a CloudWatch agent sidecar definition. To do this, append a new container called `ecs-cwagent` to your application's task definition. Replace *$REGION* with your actual Region name. Replace *$IMAGE* with the path to the latest CloudWatch container image on Amazon Elastic Container Registry. For more information, see cloudwatch-agent on Amazon ECR.

   If you want to enable the CloudWatch agent with a daemon strategy instead, see the instructions at Deploy using the daemon strategy.

   ```
   {
     "name": "ecs-cwagent",
     "image": "$IMAGE",
     "essential": true,
     "secrets": [
       {
         "name": "CW_CONFIG_CONTENT",
         "valueFrom": "ecs-cwagent"
       }
     ],
     "logConfiguration": {
   ```

```
        "logDriver": "awslogs",
        "options": {
          "awslogs-create-group": "true",
          "awslogs-group": "/ecs/ecs-cwagent",
          "awslogs-region": "$REGION",
          "awslogs-stream-prefix": "ecs"
        }
      }
    }
```

3. Append a new container `init` to your application's task definition. Replace *$IMAGE* with the latest image from the [Amazon Distro for OpenTelemetry Amazon ECR image repository](#).

```
{
  "name": "init",
  "image": "$IMAGE",
  "essential": false,
  "command": [
    "cp",
    "/javaagent.jar",
    "/otel-auto-instrumentation/javaagent.jar"
  ],
  "mountPoints": [
    {
      "sourceVolume": "opentelemetry-auto-instrumentation",
      "containerPath": "/otel-auto-instrumentation",
      "readOnly": false
    }
  ]
}
```

4. Add a dependency on the `init` container to make sure that this container finishes before your application container starts.

```
"dependsOn": [
  {
    "containerName": "init",
    "condition": "SUCCESS"
  }
]
```

5. Add the following environment variables to your application container. You must be using version 1.32.2 or later of the Amazon Distro for OpenTelemetry auto-instrumentation agent for Java.

Environment variable	Setting to enable Application Signals
`OTEL_RESOURCE_ATTRIBUTES`	Specify the following information as key-value pairs: • `service.name` sets the name of the service. This will be diplayed as the service name for your application in Application Signals dashboards. If you don't provide a value for this key, the default of `UnknownService` is used. • `deployment.environment` sets the environment that the application runs in. This will be diplayed as the **Hosted In** environment of your application in Application Signals dashboards. If you don't specify this, the default of `generic:default` is used. This attribute key is used only by Application Signals, and is converted into X-Ray trace annotations and CloudWatch metric dimensions. (Optional) To enable log correlation for Application Signals, set an additional environment variable `aws.log.group.names` to be the log group name for your application log. By doing so, the traces and metrics from your application can be correlated with the relevant log entries from the log group. For this variable, replace *`$YOUR_APPLICATION_LOG_GROUP`* with the log group names for your application. If you have multiple

Environment variable	Setting to enable Application Signals
	log groups, you can use an ampersand (&) to separate them as in this example: `aws.log.group.names=log-group-1&log-group-2` . To enable metric to log correlation, setting this current environmental variable is enough. For more information, see Enable metric to log correlation. To enable trace to log correlation, you'll also need to change the logging configuration in your application. For more information, see Enable trace to log correlation.
`OTEL_AWS_APPLICATION_SIGNALS_ENABLED`	Set to `true` to have your container start sending X-Ray traces and CloudWatch metrics to Application Signals.
`OTEL_METRICS_EXPORTER`	Set to none to disable other metrics exporters.
`OTEL_LOGS_EXPORTER`	Set to none to disable other logs exporters.
`OTEL_EXPORTER_OTLP_PROTOCOL`	Set to `http/protobuf` to send metrics and traces to Application Signals using HTTP.
`OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT`	Set to `http://localhost:4316/v1/metrics` to send metrics to the CloudWatch sidecar.
`OTEL_EXPORTER_OTLP_TRACES_ENDPOINT`	Set to `http://localhost:4316/v1/traces` to send traces to the CloudWatch sidecar.

Environment variable	Setting to enable Application Signals
OTEL_TRACES_SAMPLER	Set this to `xray` to set X-Ray as the traces sampler.
OTEL_PROPAGATORS	Set `xray` as one of the propagators.
JAVA_TOOL_OPTIONS	Set to " `-javaagent:$ AWS_ADOT_JAVA_INSTRUMENTATION_PATH` " Replace *AWS_ADOT_JAVA_INSTRUMENTATION_PATH* with the path where the Amazon Distro for OpenTelemetry Java auto-instrumentation agent is stored. For example, `/otel-auto-instrumentation/javaagent.jar`

6. Mount the volume `opentelemetry-auto-instrumentation` that you defined in step 1 of this procedure. If you don't need to enable log correlation with metrics and traces, use the following example for a Java application. If you want to enable log correlation, see the next step instead.

```
{
  "name": "my-app",
   ...
  "environment": [
    {
      "name": "OTEL_RESOURCE_ATTRIBUTES",
      "value": "service.name=$SVC_NAME"
    },
    {
      "name": "OTEL_LOGS_EXPORTER",
      "value": "none"
    },
    {
      "name": "OTEL_METRICS_EXPORTER",
      "value": "none"
    },
    {
      "name": "OTEL_EXPORTER_OTLP_PROTOCOL",
      "value": "http/protobuf"
    },
```

```
    {
      "name": "OTEL_AWS_APPLICATION_SIGNALS_ENABLED",
      "value": "true"
    },
    {
      "name": "JAVA_TOOL_OPTIONS",
      "value": " -javaagent:/otel-auto-instrumentation/javaagent.jar"
    },
    {
      "name": "OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT",
      "value": "http://localhost:4316/v1/metrics"
    },
    {
      "name": "OTEL_EXPORTER_OTLP_TRACES_ENDPOINT",
      "value": "http://localhost:4316/v1/traces"
    },
    {
      "name": "OTEL_TRACES_SAMPLER",
      "value": "xray"
    },
    {
      "name": "OTEL_PROPAGATORS",
      "value": "tracecontext,baggage,b3,xray"
    }
  ],
  "dependsOn": [
    {
      "containerName": "init",
      "condition": "SUCCESS"
    }
  ],
  "mountPoints": [
    {
      "sourceVolume": "opentelemetry-auto-instrumentation",
      "containerPath": "/otel-auto-instrumentation",
      "readOnly": false
    }
  ]
}
```

Python

Before you enable Application Signals for your Python applications, be aware of the following considerations.

- In some containerized applications, a missing PYTHONPATH environment variable can sometimes cause the application to fail to start. To resolve this, ensure that you set the PYTHONPATH environment variable to the location of your application's working directory. This is due to a known issue with OpenTelemetry auto-instrumentation. For more information about this issue, see [Python autoinstrumentation setting of PYTHONPATH is not compliant](#).
- For Django applications, there are additional required configurations, which are outlined in the [OpenTelemetry Python documentation](#).
  - Use the `--noreload` flag to prevent automatic reloading.
  - Set the `DJANGO_SETTINGS_MODULE` environment variable to the location of your Django application's `settings.py` file. This ensures that OpenTelemetry can correctly access and integrate with your Django settings.
- If you're using a WSGI server for your Python application, in addition to the following steps in this section, see [No Application Signals data for Python application that uses a WSGI server](#) for information to make Application Signals work.

To instrument your Python application on Amazon ECS with the CloudWatch agent

1. First, specify a bind mount. The volume will be used to share files across containers in the next steps. You will use this bind mount later in this procedure.

```
"volumes": [
  {
    "name": "opentelemetry-auto-instrumentation-python"
  }
]
```

2. Add a CloudWatch agent sidecar definition. To do this, append a new container called `ecs-cwagent` to your application's task definition. Replace *$REGION* with your actual Region name. Replace *$IMAGE* with the path to the latest CloudWatch container image on Amazon Elastic Container Registry. For more information, see [cloudwatch-agent](#) on Amazon ECR.

   If you want to enable the CloudWatch agent with a daemon strategy instead, see the instructions at [Deploy using the daemon strategy](#).

```
{
  "name": "ecs-cwagent",
  "image": "$IMAGE",
  "essential": true,
  "secrets": [
    {
      "name": "CW_CONFIG_CONTENT",
      "valueFrom": "ecs-cwagent"
    }
  ],
  "logConfiguration": {
    "logDriver": "awslogs",
    "options": {
      "awslogs-create-group": "true",
      "awslogs-group": "/ecs/ecs-cwagent",
      "awslogs-region": "$REGION",
      "awslogs-stream-prefix": "ecs"
    }
  }
}
```

3. Append a new container `init` to your application's task definition. Replace *$IMAGE* with the latest image from the [Amazon Distro for OpenTelemetry Amazon ECR image repository](https://gallery.ecr.aws/aws-observability/adot-autoinstrumentation-python).

```
{
    "name": "init",
    "image": "$IMAGE",
    "essential": false,
    "command": [
        "cp",
        "-a",
        "/autoinstrumentation/.",
        "/otel-auto-instrumentation-python"
    ],
    "mountPoints": [
        {
            "sourceVolume": "opentelemetry-auto-instrumentation-python",
            "containerPath": "/otel-auto-instrumentation-python",
            "readOnly": false
        }
    ]
```

```
}
```

4. Add a dependency on the `init` container to make sure that this container finishes before your application container starts.

```
"dependsOn": [
  {
    "containerName": "init",
    "condition": "SUCCESS"
  }
]
```

5. Add the following environment variables to your application container.

Environment variable	Setting to enable Application Signals
`OTEL_RESOURCE_ATTRIBUTES`	Specify the following information as key-value pairs: • `service.name` sets the name of the service. This will be diplayed as the service name for your application in Application Signals dashboards. If you don't provide a value for this key, the default of `UnknownService` is used. • `deployment.environment` sets the environment that the application runs in. This will be diplayed as the **Hosted In** environment of your application in Application Signals dashboards. If you don't specify this, the default of `generic:default` is used. This attribute key is used only by Application Signals, and is converted into X-Ray trace annotations and CloudWatch metric dimensions.

Environment variable	Setting to enable Application Signals
	(Optional) To enable log correlation for Application Signals, set an additional environment variable `aws.log.group.names` to be the log group name for your application log. By doing so, the traces and metrics from your application can be correlated with the relevant log entries from the log group. For this variable, replace *$YOUR_APPLICATION_LOG_GROUP* with the log group names for your application. If you have multiple log groups, you can use an ampersand (&) to separate them as in this example: `aws.log.group.names=log-group-1&log-group-2` . To enable metric to log correlation, setting this current environmental variable is enough. For more information, see Enable metric to log correlation. To enable trace to log correlation, you'll also need to change the logging configuration in your application. For more information, see Enable trace to log correlation.
`OTEL_AWS_APPLICATION_SIGNALS_ENABLED`	Set to `true` to have your container start sending X-Ray traces and CloudWatch metrics to Application Signals.
`OTEL_METRICS_EXPORTER`	Set to none to disable other metrics exporters.
`OTEL_EXPORTER_OTLP_PROTOCOL`	Set to `http/protobuf` to send metrics and traces to CloudWatch using HTTP.

Environment variable	Setting to enable Application Signals
`OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT`	Set to `http://127.0.0.1:4316/v1/metrics` to send metrics to the CloudWatch sidecar.
`OTEL_EXPORTER_OTLP_TRACES_ENDPOINT`	Set to `http://127.0.0.1:4316/v1/traces` to send traces to the CloudWatch sidecar.
`OTEL_TRACES_SAMPLER`	Set this to `xray` to set X-Ray as the traces sampler.
`OTEL_PROPAGATORS`	Add `xray` as one of the propagators.
`OTEL_PYTHON_DISTRO`	Set to `aws_distro` to use the ADOT Python instrumentation.
`OTEL_PYTHON_CONFIGURATOR`	Set to `aws_configurator` to use the ADOT Python configuration.
`PYTHONPATH`	Replace `$APP_PATH` with the location of the application's working directory within the container. This is required for the Python interpreter to find your application modules.
`DJANGO_SETTINGS_MODULE`	Required only for Django applications. Set it to the location of your Django application's `settings.py` file. Replace `$PATH_TO_SETTINGS`.

6. Mount the volume `opentelemetry-auto-instrumentation-python` that you defined in step 1 of this procedure. If you don't need to enable log correlation with metrics and traces, use the following example for a Python application. If you want to enable log correlation, see the next step instead.

```
{
  "name": "my-app",
```

```
  ...
  "environment": [
    {
      "name": "PYTHONPATH",
      "value": "/otel-auto-instrumentation-python/opentelemetry/instrumentation/
auto_instrumentation:$APP_PATH:/otel-auto-instrumentation-python"
    },
    {
      "name": "OTEL_EXPORTER_OTLP_PROTOCOL",
      "value": "http/protobuf"
    },
    {
      "name": "OTEL_TRACES_SAMPLER",
      "value": "xray"
    },
    {
      "name": "OTEL_TRACES_SAMPLER_ARG",
      "value": "endpoint=http://localhost:2000"
    },
    {
      "name": "OTEL_LOGS_EXPORTER",
      "value": "none"
    },
    {
      "name": "OTEL_PYTHON_DISTRO",
      "value": "aws_distro"
    },
    {
      "name": "OTEL_PYTHON_CONFIGURATOR",
      "value": "aws_configurator"
    },
    {
      "name": "OTEL_EXPORTER_OTLP_TRACES_ENDPOINT",
      "value": "http://localhost:4316/v1/traces"
    },
    {
      "name": "OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT",
      "value": "http://localhost:4316/v1/metrics"
    },
    {
      "name": "OTEL_METRICS_EXPORTER",
      "value": "none"
    },
    {
```

```
        "name": "OTEL_AWS_APPLICATION_SIGNALS_ENABLED",
        "value": "true"
      },
      {
        "name": "OTEL_RESOURCE_ATTRIBUTES",
        "value": "service.name=$SVC_NAME"
      },
      {
        "name": "DJANGO_SETTINGS_MODULE",
        "value": "$PATH_TO_SETTINGS.settings"
      }
    ],
    "mountPoints": [
      {
        "sourceVolume": "opentelemetry-auto-instrumentation-python",
        "containerPath": "/otel-auto-instrumentation-python",
        "readOnly": false
      }
    ]
}
```

7. (Optional) To enable log correlation, do the following before you mount the volume. In `OTEL_RESOURCE_ATTRIBUTES`, set an additional environment variable `aws.log.group.names` for the log groups of your application. By doing so, the traces and metrics from your application can be correlated with the relevant log entries from these log groups. For this variable, replace *$YOUR_APPLICATION_LOG_GROUP* with the log group names for your application. If you have multiple log groups, you can use an ampersand (&) to separate them as in this example: `aws.log.group.names=log-group-1&log-group-2`. To enable metric to log correlation, setting this current environmental variable is enough. For more information, see [Enable metric to log correlation](). To enable trace to log correlation, you'll also need to change the logging configuration in your application. For more information, see [Enable trace to log correlation]().

   The following is an example. To enable log correlation, use this example when you mount the volume `opentelemetry-auto-instrumentation-python` that you defined in step 1 of this procedure.

```
{
  "name": "my-app",
  ...
  "environment": [
```

```
    {
      "name": "PYTHONPATH",
      "value": "/otel-auto-instrumentation-python/opentelemetry/instrumentation/
auto_instrumentation:$APP_PATH:/otel-auto-instrumentation-python"
    },
    {
      "name": "OTEL_EXPORTER_OTLP_PROTOCOL",
      "value": "http/protobuf"
    },
    {
      "name": "OTEL_TRACES_SAMPLER",
      "value": "xray"
    },
    {
      "name": "OTEL_TRACES_SAMPLER_ARG",
      "value": "endpoint=http://localhost:2000"
    },
    {
      "name": "OTEL_LOGS_EXPORTER",
      "value": "none"
    },
    {
      "name": "OTEL_PYTHON_DISTRO",
      "value": "aws_distro"
    },
    {
      "name": "OTEL_PYTHON_CONFIGURATOR",
      "value": "aws_configurator"
    },
    {
      "name": "OTEL_EXPORTER_OTLP_TRACES_ENDPOINT",
      "value": "http://localhost:4316/v1/traces"
    },
    {
      "name": "OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT",
      "value": "http://localhost:4316/v1/metrics"
    },
    {
      "name": "OTEL_METRICS_EXPORTER",
      "value": "none"
    },
    {
      "name": "OTEL_AWS_APPLICATION_SIGNALS_ENABLED",
      "value": "true"
```

```
    },
    {
      "name": "OTEL_RESOURCE_ATTRIBUTES",
      "value":
 "aws.log.group.names=$YOUR_APPLICATION_LOG_GROUP,service.name=$SVC_NAME"
    },
    {
      "name": "DJANGO_SETTINGS_MODULE",
      "value": "$PATH_TO_SETTINGS.settings"
    }
  ],
  "mountPoints": [
    {
      "sourceVolume": "opentelemetry-auto-instrumentation-python",
      "containerPath": "/otel-auto-instrumentation-python",
      "readOnly": false
    }
  ]
}
```

.NET

To instrument your application on Amazon ECS with the CloudWatch agent

1. First, specify a bind mount. The volume will be used to share files across containers in the next steps. You will use this bind mount later in this procedure.

```
"volumes": [
  {
    "name": "opentelemetry-auto-instrumentation"
  }
]
```

2. Add a CloudWatch agent sidecar definition. To do this, append a new container called `ecs-cwagent` to your application's task definition. Replace *$REGION* with your actual Region name. Replace *$IMAGE* with the path to the latest CloudWatch container image on Amazon Elastic Container Registry. For more information, see [cloudwatch-agent](#) on Amazon ECR.

   If you want to enable the CloudWatch agent with a daemon strategy instead, see the instructions at [Deploy using the daemon strategy](#).

```
{
  "name": "ecs-cwagent",
  "image": "$IMAGE",
  "essential": true,
  "secrets": [
    {
      "name": "CW_CONFIG_CONTENT",
      "valueFrom": "ecs-cwagent"
    }
  ],
  "logConfiguration": {
    "logDriver": "awslogs",
    "options": {
      "awslogs-create-group": "true",
      "awslogs-group": "/ecs/ecs-cwagent",
      "awslogs-region": "$REGION",
      "awslogs-stream-prefix": "ecs"
    }
  }
}
```

3. Append a new container `init` to your application's task definition. Replace *$IMAGE* with the latest image from the [Amazon Distro for OpenTelemetry Amazon ECR image repository](#).

   For a Linux container instance, use the following.

```
{
  "name": "init",
  "image": "$IMAGE",
  "essential": false,
  "command": [
      "cp",
      "-a",
      "autoinstrumentation/.",
      "/otel-auto-instrumentation"
  ],
  "mountPoints": [
      {
          "sourceVolume": "opentelemetry-auto-instrumentation",
          "containerPath": "/otel-auto-instrumentation",
          "readOnly": false
```

```
        }
    ]
}
```

For a Windows Server container instance, use the following.

```
{
    "name": "init",
    "image": "$IMAGE",
    "essential": false,
    "command": [
        "CMD",
        "/c",
        "xcopy",
        "/e",
        "C:\\autoinstrumentation\\*",
        "C:\\otel-auto-instrumentation",
        "&&",
        "icacls",
        "C:\\otel-auto-instrumentation",
        "/grant",
        "*S-1-1-0:R",
        "/T"
    ],
    "mountPoints": [
        {
            "sourceVolume": "opentelemetry-auto-instrumentation",
            "containerPath": "C:\\otel-auto-instrumentation",
            "readOnly": false
        }
    ]
}
```

4. Add a dependency on the `init` container to make sure that this container finishes before your application container starts.

```
"dependsOn": [
    {
        "containerName": "init",
        "condition": "SUCCESS"
    }
]
```

5. Add the following environment variables to your application container. You must be using version 1.1.0 or later of the Amazon Distro for OpenTelemetry [auto-instrumentation agent for .NET](#).

Environment variable	Setting to enable Application Signals
`OTEL_RESOURCE_ATTRIBUTES`	Specify the following information as key-value pairs: • `service.name` sets the name of the service. This will be diplayed as the service name for your application in Application Signals dashboards. If you don't provide a value for this key, the default of `UnknownService` is used. • `deployment.environment` sets the environment that the application runs in. This will be diplayed as the **Hosted In** environment of your application in Application Signals dashboards. If you don't specify this, the default of `generic:default` is used. This attribute key is used only by Application Signals, and is converted into X-Ray trace annotations and CloudWatch metric dimensions.
`OTEL_AWS_APPLICATION_SIGNALS_ENABLED`	Set to `true` to have your container start sending X-Ray traces and CloudWatch metrics to Application Signals.
`OTEL_METRICS_EXPORTER`	Set to none to disable other metrics exporters.
`OTEL_LOGS_EXPORTER`	Set to none to disable other logs exporters.

Environment variable	Setting to enable Application Signals
`OTEL_EXPORTER_OTLP_PROTOCOL`	Set to `http/protobuf` to send metrics and traces to Application Signals using HTTP.
`OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT`	Set to `http://localhost:4316/v1/metrics` to send metrics to the CloudWatch sidecar.
`OTEL_EXPORTER_OTLP_ENDPOINT`	Set to `http://localhost:4316/` to send traces to the CloudWatch sidecar.
`OTEL_EXPORTER_OTLP_TRACES_ENDPOINT`	Set to `http://localhost:4316/v1/traces` to send traces to the CloudWatch sidecar.
`OTEL_DOTNET_AUTO_HOME`	Set to the installation location of ADOT .NET automatic instrumentation.
`OTEL_DOTNET_AUTO_PLUGINS`	Set to `AWS.Distro.OpenTelemetry.AutoInstrumentation.Plugin, AWS.Distro.OpenTelemetry.AutoInstrumentation` to enable the Application Signals plugin.
`CORECLR_ENABLE_PROFILING`	Set to 1 to enable the profiler.
`CORECLR_PROFILER`	Set to `{918728DD-259F-4A6A-AC2B-B85E1B658318}` as the CLSID of the profiler.

Environment variable	Setting to enable Application Signals
`CORECLR_PROFILER_PATH`	Set this to the path of the profiler. On Linux, set it to `${OTEL_DOTNET_AUTO_HOME}/linux-x64/OpenTelemetry.AutoInstrumentation.Native.so` On Windows Server, set it to `${OTEL_DOTNET_AUTO_HOME}/win-x64/OpenTelemetry.AutoInstrumentation.Native.dll`
`DOTNET_ADDITIONAL_DEPS`	Set this to the folder path of `${OTEL_DOTNET_AUTO_HOME}/AdditionalDeps` .
`DOTNET_SHARED_STORE`	Set this to the folder path of `${OTEL_DOTNET_AUTO_HOME}/store` .
`DOTNET_STARTUP_HOOKS`	Set this to path of the managed assembly `${OTEL_DOTNET_AUTO_HOME}/net/OpenTelemetry.AutoInstrumentation.StartupHook.dll` to run before the main application's entry point.

6. Mount the volume `opentelemetry-auto-instrumentation` that you defined in step 1 of this procedure. For Linux, use the following.

```
{
    "name": "my-app",
   ...
    "environment": [
        {
           "name": "OTEL_RESOURCE_ATTRIBUTES",
           "value": "service.name=$SVC_NAME"
       },
        {
            "name": "CORECLR_ENABLE_PROFILING",
```

```
            "value": "1"
        },
        {
            "name": "CORECLR_PROFILER",
            "value": "{918728DD-259F-4A6A-AC2B-B85E1B658318}"
        },
        {
            "name": "CORECLR_PROFILER_PATH",
            "value": "/otel-auto-instrumentation/linux-x64/
OpenTelemetry.AutoInstrumentation.Native.so"
        },
        {
            "name": "DOTNET_ADDITIONAL_DEPS",
            "value": "/otel-auto-instrumentation/AdditionalDeps"
        },
        {
            "name": "DOTNET_SHARED_STORE",
            "value": "/otel-auto-instrumentation/store"
        },
        {
            "name": "DOTNET_STARTUP_HOOKS",
            "value": "/otel-auto-instrumentation/net/
OpenTelemetry.AutoInstrumentation.StartupHook.dll"
        },
        {
            "name": "OTEL_DOTNET_AUTO_HOME",
            "value": "/otel-auto-instrumentation"
        },
        {
            "name": "OTEL_DOTNET_AUTO_PLUGINS",
            "value": "AWS.Distro.OpenTelemetry.AutoInstrumentation.Plugin,
 AWS.Distro.OpenTelemetry.AutoInstrumentation"
        },
        {
            "name": "OTEL_RESOURCE_ATTRIBUTES",
            "value": "aws.log.group.names=
$YOUR_APPLICATION_LOG_GROUP,service.name=aws-dotnet-service-name"
        },
        {
            "name": "OTEL_LOGS_EXPORTER",
            "value": "none"
        },
        {
            "name": "OTEL_METRICS_EXPORTER",
```

```
            "value": "none"
        },
        {
            "name": "OTEL_EXPORTER_OTLP_PROTOCOL",
            "value": "http/protobuf"
        },
        {
            "name": "OTEL_AWS_APPLICATION_SIGNALS_ENABLED",
            "value": "true"
        },
        {
            "name": "OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT",
            "value": "http://localhost:4316/v1/metrics"
        },
        {
            "name": "OTEL_EXPORTER_OTLP_TRACES_ENDPOINT",
            "value": "http://localhost:4316/v1/traces"
        },
        {
            "name": "OTEL_EXPORTER_OTLP_ENDPOINT",
            "value": "http://localhost:4316"
        },
        {
           "name": "OTEL_TRACES_SAMPLER",
           "value": "xray"
       },
       {
           "name": "OTEL_TRACES_SAMPLER_ARG",
           "value": "endpoint=http://localhost:2000"
       },
        {
            "name": "OTEL_PROPAGATORS",
            "value": "tracecontext,baggage,b3,xray"
        }
    ],
    "dependsOn": [
    {
      "containerName": "init",
      "condition": "SUCCESS"
    }
  ],
    "mountPoints": [
        {
            "sourceVolume": "opentelemetry-auto-instrumentation",
```

```
            "containerPath": "/otel-auto-instrumentation",
            "readOnly": false
        }
    ]
}
```

For Windows Server, use the following.

```
{
    "name": "my-app",
   ...
    "environment": [
       {
           "name": "OTEL_RESOURCE_ATTRIBUTES",
           "value": "service.name=$SVC_NAME"
       },
        {
            "name": "CORECLR_ENABLE_PROFILING",
            "value": "1"
        },
        {
            "name": "CORECLR_PROFILER",
            "value": "{918728DD-259F-4A6A-AC2B-B85E1B658318}"
        },
        {
            "name": "CORECLR_PROFILER_PATH",
            "value": "C:\\otel-auto-instrumentation\\win-x64\
\OpenTelemetry.AutoInstrumentation.Native.dll"
        },
        {
            "name": "DOTNET_ADDITIONAL_DEPS",
            "value": "C:\\otel-auto-instrumentation\\AdditionalDeps"
        },
        {
            "name": "DOTNET_SHARED_STORE",
            "value": "C:\\otel-auto-instrumentation\\store"
        },
        {
            "name": "DOTNET_STARTUP_HOOKS",
            "value": "C:\\otel-auto-instrumentation\\net\
\OpenTelemetry.AutoInstrumentation.StartupHook.dll"
        },
        {
```

```
            "name": "OTEL_DOTNET_AUTO_HOME",
            "value": "C:\\otel-auto-instrumentation"
        },
        {
            "name": "OTEL_DOTNET_AUTO_PLUGINS",
            "value": "AWS.Distro.OpenTelemetry.AutoInstrumentation.Plugin,
 AWS.Distro.OpenTelemetry.AutoInstrumentation"
        },
        {
            "name": "OTEL_RESOURCE_ATTRIBUTES",
            "value": "aws.log.group.names=
$YOUR_APPLICATION_LOG_GROUP,service.name=dotnet-service-name"
        },
        {
            "name": "OTEL_LOGS_EXPORTER",
            "value": "none"
        },
        {
            "name": "OTEL_METRICS_EXPORTER",
            "value": "none"
        },
        {
            "name": "OTEL_EXPORTER_OTLP_PROTOCOL",
            "value": "http/protobuf"
        },
        {
            "name": "OTEL_AWS_APPLICATION_SIGNALS_ENABLED",
            "value": "true"
        },
        {
            "name": "OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT",
            "value": "http://localhost:4316/v1/metrics"
        },
        {
            "name": "OTEL_EXPORTER_OTLP_TRACES_ENDPOINT",
            "value": "http://localhost:4316/v1/traces"
        },
        {
            "name": "OTEL_EXPORTER_OTLP_ENDPOINT",
            "value": "http://localhost:4316"
        },
        {
           "name": "OTEL_TRACES_SAMPLER",
           "value": "xray"
```

```
        },
        {
            "name": "OTEL_TRACES_SAMPLER_ARG",
            "value": "endpoint=http://localhost:2000"
        },
         {
             "name": "OTEL_PROPAGATORS",
             "value": "tracecontext,baggage,b3,xray"
         }
    ],
    "mountPoints": [
        {
            "sourceVolume": "opentelemetry-auto-instrumentation",
            "containerPath": "C:\\otel-auto-instrumentation",
            "readOnly": false
        }
    ],
    "dependsOn": [
        {
            "containerName": "init",
            "condition": "SUCCESS"
        }
   ]
}
```

Node.js

Note

If you are enabling Application Signals for a Node.js application with ESM, see Setting up a Node.js application with the ESM module format before you start these steps.

To instrument your application on Amazon ECS with the CloudWatch agent

1. First, specify a bind mount. The volume will be used to share files across containers in the next steps. You will use this bind mount later in this procedure.

```
"volumes": [
  {
    "name": "opentelemetry-auto-instrumentation-node"
```

```
  }
]
```

2. Add a CloudWatch agent sidecar definition. To do this, append a new container called `ecs-cwagent` to your application's task definition. Replace *$REGION* with your actual Region name. Replace *$IMAGE* with the path to the latest CloudWatch container image on Amazon Elastic Container Registry. For more information, see cloudwatch-agent on Amazon ECR.

   If you want to enable the CloudWatch agent with a daemon strategy instead, see the instructions at Deploy using the daemon strategy.

```
{
  "name": "ecs-cwagent",
  "image": "$IMAGE",
  "essential": true,
  "secrets": [
    {
      "name": "CW_CONFIG_CONTENT",
      "valueFrom": "ecs-cwagent"
    }
  ],
  "logConfiguration": {
    "logDriver": "awslogs",
    "options": {
      "awslogs-create-group": "true",
      "awslogs-group": "/ecs/ecs-cwagent",
      "awslogs-region": "$REGION",
      "awslogs-stream-prefix": "ecs"
    }
  }
}
```

3. Append a new container `init` to your application's task definition. Replace *$IMAGE* with the latest image from the Amazon Distro for OpenTelemetry Amazon ECR image repository.

```
{
  "name": "init",
  "image": "$IMAGE",
  "essential": false,
  "command": [
    "cp",
```

```
    "-a",
    "/autoinstrumentation/.",
    "/otel-auto-instrumentation-node"
  ],
  "mountPoints": [
    {
      "sourceVolume": "opentelemetry-auto-instrumentation-node",
      "containerPath": "/otel-auto-instrumentation-node",
      "readOnly": false
    }
  ],
}
```

4. Add a dependency on the `init` container to make sure that this container finishes before your application container starts.

```
"dependsOn": [
  {
    "containerName": "init",
    "condition": "SUCCESS"
  }
]
```

5. Add the following environment variables to your application container.

Environment variable	Setting to enable Application Signals
`OTEL_RESOURCE_ATTRIBUTES`	Specify the following information as key-value pairs: • `service.name` sets the name of the service. This will be diplayed as the service name for your application in Application Signals dashboards. If you don't provide a value for this key, the default of `UnknownService` is used. • `deployment.environment` sets the environment that the application runs in. This will be diplayed as the **Hosted In** environment of your applicati

Environment variable	Setting to enable Application Signals
	on in Application Signals dashboards. If you don't specify this, the default of `generic:default` is used. This attribute key is used only by Application Signals, and is converted into X-Ray trace annotations and CloudWatch metric dimensions. (Optional) To enable log correlation for Application Signals, set an additional environment variable `aws.log.group.names` to be the log group name for your application log. By doing so, the traces and metrics from your application can be correlated with the relevant log entries from the log group. For this variable, replace *$YOUR_APPLICATION_LOG_GROUP* with the log group names for your application. If you have multiple log groups, you can use an ampersand (&) to separate them as in this example: `aws.log.group.names=log-group-1&log-group-2` . To enable metric to log correlation, setting this current environmental variable is enough. For more information, see Enable metric to log correlation. To enable trace to log correlation, you'll also need to change the logging configuration in your application. For more information, see Enable trace to log correlation.

Environment variable	Setting to enable Application Signals
OTEL_AWS_APPLICATION_SIGNALS_ENABLED	Set to `true` to have your container start sending X-Ray traces and CloudWatch metrics to Application Signals.
OTEL_METRICS_EXPORTER	Set to none to disable other metrics exporters.
OTEL_LOGS_EXPORTER	Set to none to disable other logs exporters.
OTEL_EXPORTER_OTLP_PROTOCOL	Set to `http/protobuf` to send metrics and traces to Application Signals using OTLP/HTTP and protobuf.
OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT	Set to `http://localhost:4316/v1/metrics` to send metrics to the CloudWatch sidecar.
OTEL_EXPORTER_OTLP_TRACES_ENDPOINT	Set to `http://localhost:4316/v1/traces` to send traces to the CloudWatch sidecar.
OTEL_TRACES_SAMPLER	Set this to `xray` to set X-Ray as the traces sampler.
OTEL_PROPAGATORS	Set `xray` as one of the propagators.
NODE_OPTIONS	Set to `--require` *`AWS_ADOT_NODE_INSTRUMENTATION_PATH`* . Replace *`AWS_ADOT_NODE_INSTRUMENTATION_PATH`* with the path where the Amazon Distro for OpenTelemetry Node.js auto-instrumentation is stored. For example, `/otel-auto-instrumentation-node/autoinstrumentation.js`

6. Mount the volume `opentelemetry-auto-instrumentation` that you defined in step 1 of this procedure. If you don't need to enable log correlation with metrics and traces, use the following example for a Node.js application. If you want to enable log correlation, see the next step instead.

   For your Application Container, add a dependency on the `init` container to make sure that container finishes before your application container starts.

```
{
  "name": "my-app",
   ...
  "environment": [
    {
      "name": "OTEL_RESOURCE_ATTRIBUTES",
      "value": "service.name=$SVC_NAME"
    },
    {
      "name": "OTEL_LOGS_EXPORTER",
      "value": "none"
    },
    {
      "name": "OTEL_METRICS_EXPORTER",
      "value": "none"
    },
    {
      "name": "OTEL_EXPORTER_OTLP_PROTOCOL",
      "value": "http/protobuf"
    },
    {
      "name": "OTEL_AWS_APPLICATION_SIGNALS_ENABLED",
      "value": "true"
    },
    {
      "name": "OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT",
      "value": "http://localhost:4316/v1/metrics"
    },
    {
      "name": "OTEL_EXPORTER_OTLP_TRACES_ENDPOINT",
      "value": "http://localhost:4316/v1/traces"
    },
    {
      "name": "OTEL_TRACES_SAMPLER",
      "value": "xray"
```

```
    },
    {
      "name": "OTEL_TRACES_SAMPLER_ARG",
      "value": "endpoint=http://localhost:2000"
    },
    {
      "name": "NODE_OPTIONS",
      "value": "--require /otel-auto-instrumentation-node/
autoinstrumentation.js"
    }
    ],
  "mountPoints": [
    {
      "sourceVolume": "opentelemetry-auto-instrumentation-node",
      "containerPath": "/otel-auto-instrumentation-node",
      "readOnly": false
    }
  ],
  "dependsOn": [
    {
      "containerName": "init",
      "condition": "SUCCESS"
    }
  ]
}
```

7. (Optional) To enable log correlation, do the following before you mount the volume. In `OTEL_RESOURCE_ATTRIBUTES`, set an additional environment variable `aws.log.group.names` for the log groups of your application. By doing so, the traces and metrics from your application can be correlated with the relevant log entries from these log groups. For this variable, replace *`$YOUR_APPLICATION_LOG_GROUP`* with the log group names for your application. If you have multiple log groups, you can use an ampersand (&) to separate them as in this example: `aws.log.group.names=log-group-1&log-group-2`. To enable metric to log correlation, setting this current environmental variable is enough. For more information, see [Enable metric to log correlation](). To enable trace to log correlation, you'll also need to change the logging configuration in your application. For more information, see [Enable trace to log correlation]().

   The following is an example. Use this example to enable log correlation when you mount the volume `opentelemetry-auto-instrumentation` that you defined in step 1 of this procedure.

```
{
  "name": "my-app",
   ...
  "environment": [
    {
      "name": "OTEL_RESOURCE_ATTRIBUTES",
      "value":
 "aws.log.group.names=$YOUR_APPLICATION_LOG_GROUP,service.name=$SVC_NAME"
    },
    {
      "name": "OTEL_LOGS_EXPORTER",
      "value": "none"
    },
    {
      "name": "OTEL_METRICS_EXPORTER",
      "value": "none"
    },
    {
      "name": "OTEL_EXPORTER_OTLP_PROTOCOL",
      "value": "http/protobuf"
    },
    {
      "name": "OTEL_AWS_APPLICATION_SIGNALS_ENABLED",
      "value": "true"
    },
    {
      "name": "OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT",
      "value": "http://localhost:4316/v1/metrics"
    },
    {
      "name": "OTEL_EXPORTER_OTLP_TRACES_ENDPOINT",
      "value": "http://localhost:4316/v1/traces"
    },
    {
      "name": "OTEL_TRACES_SAMPLER",
      "value": "xray"
    },
    {
      "name": "OTEL_TRACES_SAMPLER_ARG",
      "value": "endpoint=http://localhost:2000"
    },
    {
      "name": "NODE_OPTIONS",
```

```
      "value": "--require /otel-auto-instrumentation-node/
autoinstrumentation.js"
    }
    ],
  "mountPoints": [
    {
      "sourceVolume": "opentelemetry-auto-instrumentation-node",
      "containerPath": "/otel-auto-instrumentation-node",
      "readOnly": false
    }
  ],
  "dependsOn": [
    {
      "containerName": "init",
      "condition": "SUCCESS"
    }
  ]
}
```

Setting up a Node.js application with the ESM module format

We provide limited support for Node.js applications with the ESM module format. For details, see the section called "Known limitations about Node.js with ESM".

For the ESM module format, using the `init` container to inject the Node.js instrumentation SDK doesn't apply. To enable Application Signals for Node.js with ESM, skip steps 1 and 3 of the previous procedure, and do the following instead.

To enable Application Signals for a Node.js application with ESM

1. Install the relevant dependencies to your Node.js application for autoinstrumentation:

```
npm install @aws/aws-distro-opentelemetry-node-autoinstrumentation
npm install @opentelemetry/instrumentation@0.54.0
```

2. In steps 5 and 6 in the previous procedure, remove the mounting of the volume `opentelemetry-auto-instrumentation-node`:

```
 "mountPoints": [
    {
```

```
            "sourceVolume": "opentelemetry-auto-instrumentation-node",
            "containerPath": "/otel-auto-instrumentation-node",
            "readOnly": false
        }
    ]
```

Replace the node options with the following.

```
{
    "name": "NODE_OPTIONS",
    "value": "--import @aws/aws-distro-opentelemetry-node-autoinstrumentation/
register --experimental-loader=@opentelemetry/instrumentation/hook.mjs"
}
```

Step 5: Deploy your application

Create a new revision of your task definition and deploy it to your application cluster. You should see three containers in the newly created task:

- `init`– A required container for initializing Application Signals.
- `ecs-cwagent`– A container running the CloudWatch agent
- *`my-app`*– This is the example application container in our documentation. In your actual workloads, this specific container might not exist or might be replaced with your own service containers.

(Optional) Step 6: Monitor your application health

Once you have enabled your applications on Amazon ECS, you can monitor your application health. For more information, see [Monitor the operational health of your applications with Application Signals](#).

Deploy using the daemon strategy

Step 1: Enable Application Signals in your account

You must first enable Application Signals in your account. If you haven't, see [Enable Application Signals in your account](#).

Step 2: Create IAM roles

You must create an IAM role. If you already have created this role, you might need to add permissions to it.

- **ECS task role—** Containers use this role to run. The permissions should be whatever your applications need, plus **CloudWatchAgentServerPolicy**.

For more information about creating IAM roles, see Creating IAM Roles.

Step 3: Prepare CloudWatch agent configuration

First, prepare the agent configuration with Application Signals enabled. To do this, create a local file named `/tmp/ecs-cwagent.json`.

```
{
  "traces": {
    "traces_collected": {
      "application_signals": {}
    }
  },
  "logs": {
    "metrics_collected": {
      "application_signals": {}
    }
  }
}
```

Then upload this configuration to the SSM Parameter Store. To do this, enter the following command. In the file, replace *$REGION* with your actual Region name.

```
aws ssm put-parameter \
--name "ecs-cwagent" \
--type "String" \
--value "`cat /tmp/ecs-cwagent.json`" \
--region "$REGION"
```

Step 4: Deploy the CloudWatch agent daemon service

Create the following task definition and deploy it to your application cluster. Replace *$REGION* with your actual Region name. Replace *$TASK_ROLE_ARN* and *$EXECUTION_ROLE_ARN* with the

IAM roles you prepared in Step 2: Create IAM roles. Replace *$IMAGE* with the path to the latest CloudWatch container image on Amazon Elastic Container Registry. For more information, see cloudwatch-agent on Amazon ECR.

Note

The daemon service exposes two ports on the host, with 4316 used as endpoint for receiving metrics and traces and 2000 as the CloudWatch trace sampler endpoint. This setup allows the agent to collect and transmit telemetry data from all application tasks running on the host. Ensure that these ports are not used by other services on the host to avoid conflicts.

```
{
  "family": "ecs-cwagent-daemon",
  "taskRoleArn": "$TASK_ROLE_ARN",
  "executionRoleArn": "$EXECUTION_ROLE_ARN",
  "networkMode": "bridge",
  "containerDefinitions": [
    {
      "name": "ecs-cwagent",
      "image": "$IMAGE",
      "essential": true,
      "portMappings": [
        {
          "containerPort": 4316,
          "hostPort": 4316
        },
        {
          "containerPort": 2000,
          "hostPort": 2000
        }
      ],
      "secrets": [
        {
          "name": "CW_CONFIG_CONTENT",
          "valueFrom": "ecs-cwagent"
        }
      ],
      "logConfiguration": {
        "logDriver": "awslogs",
```

```
        "options": {
          "awslogs-create-group": "true",
          "awslogs-group": "/ecs/ecs-cwagent",
          "awslogs-region": "$REGION",
          "awslogs-stream-prefix": "ecs"
        }
      }
    }
  ],
  "requiresCompatibilities": [
    "EC2"
  ],
  "cpu": "128",
  "memory": "64"
}
```

Step 5: Instrument your application

The next step is to instrument your application for Application Signals.

Java

To instrument your application on Amazon ECS with the CloudWatch agent

1. First, specify a bind mount. The volume will be used to share files across containers in the next steps. You will use this bind mount later in this procedure.

```
"volumes": [
  {
    "name": "opentelemetry-auto-instrumentation"
  }
]
```

2. Append a new container `init` to your application's task definition. Replace *$IMAGE* with the latest image from the [Amazon Distro for OpenTelemetry Amazon ECR image repository](#).

```
{
  "name": "init",
  "image": "$IMAGE",
  "essential": false,
  "command": [
    "cp",
```

```
    "/javaagent.jar",
    "/otel-auto-instrumentation/javaagent.jar"
  ],
  "mountPoints": [
    {
      "sourceVolume": "opentelemetry-auto-instrumentation",
      "containerPath": "/otel-auto-instrumentation",
      "readOnly": false
    }
  ]
}
```

3. Add a dependency on the `init` container to make sure that this container finishes before your application container starts.

```
"dependsOn": [
  {
    "containerName": "init",
    "condition": "SUCCESS"
  }
]
```

4. Add the following environment variables to your application container. You must be using version 1.32.2 or later of the Amazon Distro for OpenTelemetry [auto-instrumentation agent for Java](https://).

Environment variable	Setting to enable Application Signals
`OTEL_RESOURCE_ATTRIBUTES`	Specify the following information as key-value pairs: • `service.name` sets the name of the service. This will be diplayed as the service name for your application in Application Signals dashboards. If you don't provide a value for this key, the default of `UnknownService` is used. • `deployment.environment` sets the environment that the application runs in. This will be diplayed as the

Environment variable	Setting to enable Application Signals
	Hosted In environment of your application in Application Signals dashboards. If you don't specify this, the default of `generic:default` is used. This attribute key is used only by Application Signals, and is converted into X-Ray trace annotations and CloudWatch metric dimensions. (Optional) To enable log correlation for Application Signals, set an additional environment variable `aws.log.group.names` to be the log group name for your application log. By doing so, the traces and metrics from your application can be correlated with the relevant log entries from the log group. For this variable, replace *`$YOUR_APPLICATION_LOG_GROUP`* with the log group names for your application. If you have multiple log groups, you can use an ampersand (&) to separate them as in this example: `aws.log.group.names=log-group-1&log-group-2` . To enable metric to log correlation, setting this current environmental variable is enough. For more information, see [Enable metric to log correlation](). To enable trace to log correlation, you'll also need to change the logging configuration in your application. For more information, see [Enable trace to log correlation]().

Environment variable	Setting to enable Application Signals
`OTEL_AWS_APPLICATION_SIGNALS_ENABLED`	Set to `true` to have your container start sending X-Ray traces and CloudWatch metrics to Application Signals.
`OTEL_METRICS_EXPORTER`	Set to none to disable other metrics exporters.
`OTEL_LOGS_EXPORTER`	Set to none to disable other logs exporters.
`OTEL_EXPORTER_OTLP_PROTOCOL`	Set to `http/protobuf` to send metrics and traces to Application Signals using HTTP.
`OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT`	Sends metrics to the CloudWatch daemon container. • For applications running in `host` mode, set this to `http://localhost:4316/v1/metrics` . • For applications running in `bridge` mode or `awsvpc` mode, set this to http://*CW_CONTAINER_IP* :4316/v1/metrics, where *CW_CONTAINER_IP* is the private IP address of the EC2 container instance. You can retrieve this address from the Instance Metadata Service (IMDS).

Environment variable	Setting to enable Application Signals
OTEL_EXPORTER_OTLP_TRACES_ENDPOINT	Sends traces to the CloudWatch daemon container. • For applications running in `host` mode, set this to `http://localhost:4316/v1/traces` . • For applications running in `bridge` mode or `awsvpc` mode, set this to http://*CW_CONTAINER_IP* :4316/v1/traces, where *CW_CONTAINER_IP* is the private IP address of the EC2 container instance. You can retrieve this address from the Instance Metadata Service (IMDS).
OTEL_TRACES_SAMPLER	Set this to `xray` to set X-Ray as the traces sampler.
OTEL_PROPAGATORS	Set `xray` as one of the propagators.
JAVA_TOOL_OPTIONS	Set to " `-javaagent:$` *AWS_ADOT_JAVA_INSTRUMENTATION_PATH* " Replace *AWS_ADOT_JAVA_INSTRUMENTATION_PATH* with the path where the Amazon Distro for OpenTelemetry Java auto-instrumentation agent is stored. For example, `/otel-auto-instrumentation/javaagent.jar`

5. Mount the volume `opentelemetry-auto-instrumentation` that you defined in step 1 of this procedure. If you don't need to enable log correlation with metrics and traces, use the following example for a Java application. If you want to enable log correlation, see the next step instead.

```
{
  "name": "my-app",
```

```
...
"environment": [
  {
    "name": "OTEL_RESOURCE_ATTRIBUTES",
    "value": "service.name=$SVC_NAME"
  },
  {
    "name": "OTEL_LOGS_EXPORTER",
    "value": "none"
  },
  {
    "name": "OTEL_METRICS_EXPORTER",
    "value": "none"
  },
  {
    "name": "OTEL_EXPORTER_OTLP_PROTOCOL",
    "value": "http/protobuf"
  },
  {
    "name": "OTEL_AWS_APPLICATION_SIGNALS_ENABLED",
    "value": "true"
  },
  {
    "name": "JAVA_TOOL_OPTIONS",
    "value": " -javaagent:/otel-auto-instrumentation/javaagent.jar"
  },
  {
    "name": "OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT",
    "value": "http://CW_CONTAINER_IP:4316/v1/metrics"
  },
  {
    "name": "OTEL_EXPORTER_OTLP_TRACES_ENDPOINT",
    "value": "http://CW_CONTAINER_IP:4316/v1/traces"
  },
  {
    "name": "OTEL_TRACES_SAMPLER",
    "value": "xray"
  },
  {
    "name": "OTEL_PROPAGATORS",
    "value": "tracecontext,baggage,b3,xray"
  }
],
  "dependsOn": [
```

```
    {
      "containerName": "init",
      "condition": "SUCCESS"
    }
  ],
  "mountPoints": [
    {
      "sourceVolume": "opentelemetry-auto-instrumentation",
      "containerPath": "/otel-auto-instrumentation",
      "readOnly": false
    }
  ]
}
```

Python

Before you enable Application Signals for your Python applications, be aware of the following considerations.

- In some containerized applications, a missing `PYTHONPATH` environment variable can sometimes cause the application to fail to start. To resolve this, ensure that you set the `PYTHONPATH` environment variable to the location of your application's working directory. This is due to a known issue with OpenTelemetry auto-instrumentation. For more information about this issue, see [Python autoinstrumentation setting of PYTHONPATH is not compliant].
- For Django applications, there are additional required configurations, which are outlined in the [OpenTelemetry Python documentation].
  - Use the `--noreload` flag to prevent automatic reloading.
  - Set the `DJANGO_SETTINGS_MODULE` environment variable to the location of your Django application's `settings.py` file. This ensures that OpenTelemetry can correctly access and integrate with your Django settings.
- If you're using a WSGI server for your Python application, in addition to the following steps in this section, see [No Application Signals data for Python application that uses a WSGI server] for information to make Application Signals work.

To instrument your Python application on Amazon ECS with the CloudWatch agent

1. First, specify a bind mount. The volume will be used to share files across containers in the next steps. You will use this bind mount later in this procedure.

```
"volumes": [
  {
    "name": "opentelemetry-auto-instrumentation-python"
  }
]
```

2. Append a new container `init` to your application's task definition. Replace *$IMAGE* with the latest image from the [Amazon Distro for OpenTelemetry Amazon ECR image repository](https://gallery.ecr.aws/).

```
{
    "name": "init",
    "image": "$IMAGE",
    "essential": false,
    "command": [
        "cp",
        "-a",
        "/autoinstrumentation/.",
        "/otel-auto-instrumentation-python"
    ],
    "mountPoints": [
        {
            "sourceVolume": "opentelemetry-auto-instrumentation-python",
            "containerPath": "/otel-auto-instrumentation-python",
            "readOnly": false
        }
    ]
}
```

3. Add a dependency on the `init` container to make sure that this container finishes before your application container starts.

```
"dependsOn": [
  {
    "containerName": "init",
    "condition": "SUCCESS"
  }
]
```

4. Add the following environment variables to your application container.

Environment variable	Setting to enable Application Signals
OTEL_RESOURCE_ATTRIBUTES	Specify the following information as key-value pairs: • service.name sets the name of the service. This will be diplayed as the service name for your application in Application Signals dashboards. If you don't provide a value for this key, the default of UnknownService is used. • deployment.environment sets the environment that the application runs in. This will be diplayed as the **Hosted In** environment of your application in Application Signals dashboards. If you don't specify this, the default of generic:default is used. This attribute key is used only by Application Signals, and is converted into X-Ray trace annotations and CloudWatch metric dimensions. (Optional) To enable log correlation for Application Signals, set an additional environment variable aws.log.group.names to be the log group name for your application log. By doing so, the traces and metrics from your application can be correlated with the relevant log entries from the log group. For this variable, replace *$YOUR_APPLICATION_LOG_GROUP* with the log group names for your application. If you have multiple

Environment variable	Setting to enable Application Signals
	log groups, you can use an ampersand (&) to separate them as in this example: `aws.log.group.names=log-group-1&log-group-2` . To enable metric to log correlation, setting this current environmental variable is enough. For more information, see Enable metric to log correlation. To enable trace to log correlation, you'll also need to change the logging configuration in your application. For more information, see Enable trace to log correlation.
`OTEL_AWS_APPLICATION_SIGNALS_ENABLED`	Set to `true` to have your container start sending X-Ray traces and CloudWatch metrics to Application Signals.
`OTEL_METRICS_EXPORTER`	Set to none to disable other metrics exporters.
`OTEL_EXPORTER_OTLP_PROTOCOL`	Set to `http/protobuf` to send metrics and traces to CloudWatch using HTTP.

Environment variable	Setting to enable Application Signals
`OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT`	Sends metrics to the CloudWatch daemon container. • For applications running in `host` mode, set this to `http://localhost:4316/v1/metrics` . • For applications running in `bridge` mode or `awsvpc` mode, set this to http://*`CW_CONTAINER_IP`* :4316/v1/metrics, where *`CW_CONTAINER_IP`* is the private IP address of the EC2 container instance. You can retrieve this address from the Instance Metadata Service (IMDS).
`OTEL_EXPORTER_OTLP_TRACES_ENDPOINT`	Sends traces to the CloudWatch daemon container. • For applications running in `host` mode, set this to `http://localhost:4316/v1/traces` . • For applications running in `bridge` mode or `awsvpc` mode, set this to http://*`CW_CONTAINER_IP`* :4316/v1/traces, where *`CW_CONTAINER_IP`* is the private IP address of the EC2 container instance. You can retrieve this address from the Instance Metadata Service (IMDS).
`OTEL_TRACES_SAMPLER`	Set this to `xray` to set X-Ray as the traces sampler.

Environment variable	Setting to enable Application Signals
OTEL_TRACES_SAMPLER_ARG	Sets the traces sampler endpoint. • For applications running in `host` mode, set this to `http://localhost:2000` . • For applications running in `bridge` mode or `awsvpc` mode, set this to `http://CW_CONTAINER_IP :2000`, where *CW_CONTAINER_IP* is the private IP address of the EC2 container instance. You can retrieve this address from the Instance Metadata Service (IMDS).
OTEL_PROPAGATORS	Add `xray` as one of the propagators.
OTEL_PYTHON_DISTRO	Set to `aws_distro` to use the ADOT Python instrumentation.
OTEL_PYTHON_CONFIGURATOR	Set to `aws_configuration` to use the ADOT Python configuration.
PYTHONPATH	Replace `$APP_PATH` with the location of the application's working directory within the container. This is required for the Python interpreter to find your application modules.
DJANGO_SETTINGS_MODULE	Required only for Django applications. Set it to the location of your Django application's `settings.py` file. Replace `$PATH_TO_SETTINGS` .

5. Mount the volume `opentelemetry-auto-instrumentation-python` that you defined in step 1 of this procedure. If you don't need to enable log correlation with metrics and

traces, use the following example for a Python application. If you want to enable log correlation, see the next step instead.

```
{
  "name": "my-app",
  ...
  "environment": [
    {
      "name": "PYTHONPATH",
      "value": "/otel-auto-instrumentation-python/opentelemetry/instrumentation/
auto_instrumentation:$APP_PATH:/otel-auto-instrumentation-python"
    },
    {
      "name": "OTEL_EXPORTER_OTLP_PROTOCOL",
      "value": "http/protobuf"
    },
    {
      "name": "OTEL_TRACES_SAMPLER",
      "value": "xray"
    },
    {
      "name": "OTEL_TRACES_SAMPLER_ARG",
      "value": "endpoint=http://CW_CONTAINER_IP:2000"
    },
    {
      "name": "OTEL_LOGS_EXPORTER",
      "value": "none"
    },
    {
      "name": "OTEL_PYTHON_DISTRO",
      "value": "aws_distro"
    },
    {
      "name": "OTEL_PYTHON_CONFIGURATOR",
      "value": "aws_configurator"
    },
    {
      "name": "OTEL_EXPORTER_OTLP_TRACES_ENDPOINT",
      "value": "http://CW_CONTAINER_IP:4316/v1/traces"
    },
    {
      "name": "OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT",
      "value": "http://CW_CONTAINER_IP:4316/v1/metrics"
```

```
      },
      {
        "name": "OTEL_METRICS_EXPORTER",
        "value": "none"
      },
      {
        "name": "OTEL_AWS_APPLICATION_SIGNALS_ENABLED",
        "value": "true"
      },
      {
        "name": "OTEL_RESOURCE_ATTRIBUTES",
        "value": "service.name=$SVC_NAME"
      },
      {
        "name": "DJANGO_SETTINGS_MODULE",
        "value": "$PATH_TO_SETTINGS.settings"
      }
    ],
    "dependsOn": [
      {
        "containerName": "init",
        "condition": "SUCCESS"
      }
    ],
    "mountPoints": [
      {
        "sourceVolume": "opentelemetry-auto-instrumentation-python",
        "containerPath": "/otel-auto-instrumentation-python",
        "readOnly": false
      }
    ]
}
```

6. (Optional) To enable log correlation, do the following before you mount the volume. In `OTEL_RESOURCE_ATTRIBUTES`, set an additional environment variable `aws.log.group.names` for the log groups of your application. By doing so, the traces and metrics from your application can be correlated with the relevant log entries from these log groups. For this variable, replace *`$YOUR_APPLICATION_LOG_GROUP`* with the log group names for your application. If you have multiple log groups, you can use an ampersand (&) to separate them as in this example: `aws.log.group.names=log-group-1&log-group-2`. To enable metric to log correlation, setting this current environmental variable is enough. For more information, see Enable metric to log correlation. To enable trace to

log correlation, you'll also need to change the logging configuration in your application. For more information, see Enable trace to log correlation.

The following is an example. To enable log correlation, use this example when you mount the volume `opentelemetry-auto-instrumentation-python` that you defined in step 1 of this procedure.

```
{
  "name": "my-app",
  ...
  "environment": [
    {
      "name": "PYTHONPATH",
      "value": "/otel-auto-instrumentation-python/opentelemetry/instrumentation/
auto_instrumentation:$APP_PATH:/otel-auto-instrumentation-python"
    },
    {
      "name": "OTEL_EXPORTER_OTLP_PROTOCOL",
      "value": "http/protobuf"
    },
    {
      "name": "OTEL_TRACES_SAMPLER",
      "value": "xray"
    },
    {
      "name": "OTEL_TRACES_SAMPLER_ARG",
      "value": "endpoint=http://CW_CONTAINER_IP:2000"
    },
    {
      "name": "OTEL_LOGS_EXPORTER",
      "value": "none"
    },
    {
      "name": "OTEL_PYTHON_DISTRO",
      "value": "aws_distro"
    },
    {
      "name": "OTEL_PYTHON_CONFIGURATOR",
      "value": "aws_configurator"
    },
    {
      "name": "OTEL_EXPORTER_OTLP_TRACES_ENDPOINT",
      "value": "http://CW_CONTAINER_IP:4316/v1/traces"
```

```
    },
    {
      "name": "OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT",
      "value": "http://CW_CONTAINER_IP:4316/v1/metrics"
    },
    {
      "name": "OTEL_METRICS_EXPORTER",
      "value": "none"
    },
    {
      "name": "OTEL_AWS_APPLICATION_SIGNALS_ENABLED",
      "value": "true"
    },
    {
      "name": "OTEL_RESOURCE_ATTRIBUTES",
      "value":
 "aws.log.group.names=$YOUR_APPLICATION_LOG_GROUP,service.name=$SVC_NAME"
    },
    {
      "name": "DJANGO_SETTINGS_MODULE",
      "value": "$PATH_TO_SETTINGS.settings"
    }
  ],
  "dependsOn": [
    {
      "containerName": "init",
      "condition": "SUCCESS"
    }
  ],
  "mountPoints": [
    {
      "sourceVolume": "opentelemetry-auto-instrumentation-python",
      "containerPath": "/otel-auto-instrumentation-python",
      "readOnly": false
    }
  ]
}
```

.NET

To instrument your application on Amazon ECS with the CloudWatch agent

1. First, specify a bind mount. The volume will be used to share files across containers in the next steps. You will use this bind mount later in this procedure.

```
"volumes": [
  {
    "name": "opentelemetry-auto-instrumentation"
  }
]
```

2. Append a new container `init` to your application's task definition. Replace *$IMAGE* with the latest image from the [Amazon Distro for OpenTelemetry Amazon ECR image repository](https://gallery.ecr.aws/aws-observability/adot-autoinstrumentation-dotnet).

   For a Linux container instance, use the following.

```
{
  "name": "init",
  "image": "$IMAGE",
  "essential": false,
  "command": [
      "cp",
      "-a",
      "autoinstrumentation/.",
      "/otel-auto-instrumentation"
  ],
  "mountPoints": [
      {
          "sourceVolume": "opentelemetry-auto-instrumentation",
          "containerPath": "/otel-auto-instrumentation",
          "readOnly": false
      }
  ]
}
```

   For a Windows Server container instance, use the following.

```
{
  "name": "init",
```

```
    "image": "$IMAGE",
    "essential": false,
    "command": [
        "CMD",
        "/c",
        "xcopy",
        "/e",
        "C:\\autoinstrumentation\\*",
        "C:\\otel-auto-instrumentation",
        "&&",
        "icacls",
        "C:\\otel-auto-instrumentation",
        "/grant",
        "*S-1-1-0:R",
        "/T"
    ],
    "mountPoints": [
        {
            "sourceVolume": "opentelemetry-auto-instrumentation",
            "containerPath": "C:\\otel-auto-instrumentation",
            "readOnly": false
        }
    ]
}
```

3. Add a dependency on the `init` container to make sure that container finishes before your application container starts.

```
"dependsOn": [
    {
        "containerName": "init",
        "condition": "SUCCESS"
    }
]
```

4. Add the following environment variables to your application container. You must be using version 1.1.0 or later of the Amazon Distro for OpenTelemetry auto-instrumentation agent for .NET.

Environment variable	Setting to enable Application Signals
`OTEL_RESOURCE_ATTRIBUTES`	Specify the following information as key-value pairs: • `service.name` sets the name of the service. This will be diplayed as the service name for your application in Application Signals dashboards. If you don't provide a value for this key, the default of `UnknownService` is used. • `deployment.environment` sets the environment that the application runs in. This will be diplayed as the **Hosted In** environment of your application in Application Signals dashboards. If you don't specify this, the default of `generic:default` is used. This attribute key is used only by Application Signals, and is converted into X-Ray trace annotations and CloudWatch metric dimensions.
`OTEL_AWS_APPLICATION_SIGNALS_ENABLED`	Set to `true` to have your container start sending X-Ray traces and CloudWatch metrics to Application Signals.
`OTEL_METRICS_EXPORTER`	Set to none to disable other metrics exporters.
`OTEL_LOGS_EXPORTER`	Set to none to disable other logs exporters.

Environment variable	Setting to enable Application Signals
`OTEL_EXPORTER_OTLP_PROTOCOL`	Set to `http/protobuf` to send metrics and traces to Application Signals using HTTP.
`OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT`	Sends metrics to the CloudWatch daemon container. • For applications running in `host` mode, set this to `http://localhost:4316/v1/metrics`. • For applications running in `bridge` mode or `awsvpc` mode, set this to http://*CW_CONTAINER_IP*:4316/v1/metrics, where *CW_CONTAINER_IP* is the private IP address of the EC2 container instance. You can retrieve this address from the Instance Metadata Service (IMDS).
`OTEL_EXPORTER_OTLP_ENDPOINT`	Sends traces to the CloudWatch daemon container. • For applications running in `host` mode, set this to `http://localhost:4316`. • For applications running in `bridge` mode or `awsvpc` mode, set this to http://*CW_CONTAINER_IP*:4316, where *CW_CONTAINER_IP* is the private IP address of the EC2 container instance. You can retrieve this address from the Instance Metadata Service (IMDS).

Environment variable	Setting to enable Application Signals
`OTEL_EXPORTER_OTLP_TRACES_ENDPOINT`	Sends traces to the CloudWatch daemon container. • For applications running in `host` mode, set this to `http://localhost:4316/v1/traces` . • For applications running in `bridge` mode or `awsvpc` mode, set this to http://*CW_CONTAINER_IP* :4316/v1/traces, where *CW_CONTAINER_IP* is the private IP address of the EC2 container instance. You can retrieve this address from the Instance Metadata Service (IMDS).
`OTEL_TRACES_SAMPLER_ARG`	Sets the traces sampler endpoint. • For applications running in `host` mode, set this to `http://localhost:2000` . • For applications running in `bridge` mode or `awsvpc` mode, set this to `http://`*CW_CONTAINER_IP* `:2000`, where *CW_CONTAINER_IP* is the private IP address of the EC2 container instance. You can retrieve this address from the Instance Metadata Service (IMDS).
`OTEL_DOTNET_AUTO_HOME`	Set to the installation location of ADOT .NET automatic instrumentation.

Environment variable	Setting to enable Application Signals
OTEL_DOTNET_AUTO_PLUGINS	Set to AWS.Distro.OpenTelemetry.AutoInstrumentation.Plugin, AWS.Distro.OpenTelemetry.AutoInstrumentation to enable the Application Signals plugin.
CORECLR_ENABLE_PROFILING	Set to 1 to enable the profiler.
CORECLR_PROFILER	Set to {918728DD-259F-4A6A-AC2B-B85E1B658318} as the CLSID of the profiler.
CORECLR_PROFILER_PATH	Set this to the path of the profiler. On Linux, set it to ${OTEL_DOTNET_AUTO_HOME}/linux-x64/OpenTelemetry.AutoInstrumentation.Native.so On Windows Server, set it to ${OTEL_DOTNET_AUTO_HOME}/win-x64/OpenTelemetry.AutoInstrumentation.Native.dll
DOTNET_ADDITIONAL_DEPS	Set this to the folder path of ${OTEL_DOTNET_AUTO_HOME}/AdditionalDeps .
DOTNET_SHARED_STORE	Set this to the folder path of ${OTEL_DOTNET_AUTO_HOME}/store .
DOTNET_STARTUP_HOOKS	Set this to path of the managed assembly ${OTEL_DOTNET_AUTO_HOME}/net/OpenTelemetry.AutoInstrumentation.StartupHook.dll to run before the main application's entry point.

5. Mount the volume `opentelemetry-auto-instrumentation` that you defined in step 1 of this procedure. For Linux, use the following.

```
{
    "name": "my-app",
   ...
    "environment": [
        {
           "name": "OTEL_RESOURCE_ATTRIBUTES",
           "value": "service.name=$SVC_NAME"
       },
        {
            "name": "CORECLR_ENABLE_PROFILING",
            "value": "1"
        },
        {
            "name": "CORECLR_PROFILER",
            "value": "{918728DD-259F-4A6A-AC2B-B85E1B658318}"
        },
        {
            "name": "CORECLR_PROFILER_PATH",
            "value": "/otel-auto-instrumentation/linux-x64/
OpenTelemetry.AutoInstrumentation.Native.so"
        },
        {
            "name": "DOTNET_ADDITIONAL_DEPS",
            "value": "/otel-auto-instrumentation/AdditionalDeps"
        },
        {
            "name": "DOTNET_SHARED_STORE",
            "value": "/otel-auto-instrumentation/store"
        },
        {
            "name": "DOTNET_STARTUP_HOOKS",
            "value": "/otel-auto-instrumentation/net/
OpenTelemetry.AutoInstrumentation.StartupHook.dll"
        },
        {
            "name": "OTEL_DOTNET_AUTO_HOME",
            "value": "/otel-auto-instrumentation"
        },
        {
            "name": "OTEL_DOTNET_AUTO_PLUGINS",
```

```
            "value": "AWS.Distro.OpenTelemetry.AutoInstrumentation.Plugin,
 AWS.Distro.OpenTelemetry.AutoInstrumentation"
        },
        {
            "name": "OTEL_RESOURCE_ATTRIBUTES",
            "value":
 "aws.log.group.names=$YOUR_APPLICATION_LOG_GROUP,service.name=dotnet-service-
name"
        },
        {
            "name": "OTEL_LOGS_EXPORTER",
            "value": "none"
        },
        {
            "name": "OTEL_METRICS_EXPORTER",
            "value": "none"
        },
        {
            "name": "OTEL_EXPORTER_OTLP_PROTOCOL",
            "value": "http/protobuf"
        },
        {
            "name": "OTEL_AWS_APPLICATION_SIGNALS_ENABLED",
            "value": "true"
        },
        {
            "name": "OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT",
            "value": "http://localhost:4316/v1/metrics"
        },
        {
            "name": "OTEL_EXPORTER_OTLP_TRACES_ENDPOINT",
            "value": "http://CW_CONTAINER_IP:4316/v1/traces"
        },
        {
            "name": "OTEL_EXPORTER_OTLP_ENDPOINT",
            "value": "http://CW_CONTAINER_IP:4316"
        },
        {
           "name": "OTEL_TRACES_SAMPLER",
           "value": "xray"
       },
       {
           "name": "OTEL_TRACES_SAMPLER_ARG",
           "value": "endpoint=http://CW_CONTAINER_IP:2000"
```

```
        },
         {
             "name": "OTEL_PROPAGATORS",
             "value": "tracecontext,baggage,b3,xray"
         }
    ],
      "dependsOn": [
    {
      "containerName": "init",
      "condition": "SUCCESS"
    }
  ],
    "mountPoints": [
        {
            "sourceVolume": "opentelemetry-auto-instrumentation",
            "containerPath": "/otel-auto-instrumentation",
            "readOnly": false
        }
    ],
    "dependsOn": [
        {
            "containerName": "init",
            "condition": "SUCCESS"
        }
   ]
}
```

For Windows Server, use the following.

```
{
    "name": "my-app",
   ...
    "environment": [
       {
           "name": "OTEL_RESOURCE_ATTRIBUTES",
           "value": "service.name=$SVC_NAME"
       },
        {
            "name": "CORECLR_ENABLE_PROFILING",
            "value": "1"
        },
        {
            "name": "CORECLR_PROFILER",
```

```
            "value": "{918728DD-259F-4A6A-AC2B-B85E1B658318}"
        },
        {
            "name": "CORECLR_PROFILER_PATH",
            "value": "C:\\otel-auto-instrumentation\\win-x64\
\OpenTelemetry.AutoInstrumentation.Native.dll"
        },
        {
            "name": "DOTNET_ADDITIONAL_DEPS",
            "value": "C:\\otel-auto-instrumentation\\AdditionalDeps"
        },
        {
            "name": "DOTNET_SHARED_STORE",
            "value": "C:\\otel-auto-instrumentation\\store"
        },
        {
            "name": "DOTNET_STARTUP_HOOKS",
            "value": "C:\\otel-auto-instrumentation\\net\
\OpenTelemetry.AutoInstrumentation.StartupHook.dll"
        },
        {
            "name": "OTEL_DOTNET_AUTO_HOME",
            "value": "C:\\otel-auto-instrumentation"
        },
        {
            "name": "OTEL_DOTNET_AUTO_PLUGINS",
            "value": "AWS.Distro.OpenTelemetry.AutoInstrumentation.Plugin,
 AWS.Distro.OpenTelemetry.AutoInstrumentation"
        },
        {
            "name": "OTEL_RESOURCE_ATTRIBUTES",
            "value":
 "aws.log.group.names=$YOUR_APPLICATION_LOG_GROUP,service.name=dotnet-service-
name"
        },
        {
            "name": "OTEL_LOGS_EXPORTER",
            "value": "none"
        },
        {
            "name": "OTEL_METRICS_EXPORTER",
            "value": "none"
        },
        {
```

```
                "name": "OTEL_EXPORTER_OTLP_PROTOCOL",
                "value": "http/protobuf"
            },
            {
                "name": "OTEL_AWS_APPLICATION_SIGNALS_ENABLED",
                "value": "true"
            },
            {
                "name": "OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT",
                "value": "http://CW_CONTAINER_IP:4316/v1/metrics"
            },
            {
                "name": "OTEL_EXPORTER_OTLP_TRACES_ENDPOINT",
                "value": "http://CW_CONTAINER_IP:4316/v1/traces"
            },
            {
                "name": "OTEL_EXPORTER_OTLP_ENDPOINT",
                "value": "http://CW_CONTAINER_IP:4316"
            },
            {
               "name": "OTEL_TRACES_SAMPLER",
               "value": "xray"
           },
           {
               "name": "OTEL_TRACES_SAMPLER_ARG",
               "value": "endpoint=http://CW_CONTAINER_IP:2000"
           },
            {
                "name": "OTEL_PROPAGATORS",
                "value": "tracecontext,baggage,b3,xray"
            }
        ],
        "mountPoints": [
            {
                "sourceVolume": "opentelemetry-auto-instrumentation",
                "containerPath": "C:\\otel-auto-instrumentation",
                "readOnly": false
            }
        ],
        "dependsOn": [
            {
                "containerName": "init",
                "condition": "SUCCESS"
            }
```

```
    ]
}
```

Node.js

> **Note**
>
> If you are enabling Application Signals for a Node.js application with ESM, see Setting up a Node.js application with the ESM module format before you start these steps.

To instrument your application on Amazon ECS with the CloudWatch agent

1. First, specify a bind mount. The volume will be used to share files across containers in the next steps. You will use this bind mount later in this procedure.

```
"volumes": [
  {
    "name": "opentelemetry-auto-instrumentation-node"
  }
]
```

2. Append a new container `init` to your application's task definition. Replace *$IMAGE* with the latest image from the Amazon Distro for OpenTelemetry Amazon ECR image repository.

```
{
  "name": "init",
  "image": "$IMAGE",
  "essential": false,
  "command": [
    "cp",
    "-a",
    "/autoinstrumentation/.",
    "/otel-auto-instrumentation-node"
  ],
  "mountPoints": [
    {
      "sourceVolume": "opentelemetry-auto-instrumentation-node",
      "containerPath": "/otel-auto-instrumentation-node",
      "readOnly": false
```

```
    }
  ],
}
```

3. Add a dependency on the `init` container to make sure that this container finishes before your application container starts.

```
"dependsOn": [
  {
    "containerName": "init",
    "condition": "SUCCESS"
  }
]
```

4. Add the following environment variables to your application container.

Environment variable	Setting to enable Application Signals
`OTEL_RESOURCE_ATTRIBUTES`	Specify the following information as key-value pairs: • `service.name` sets the name of the service. This will be diplayed as the service name for your application in Application Signals dashboards. If you don't provide a value for this key, the default of `UnknownService` is used. • `deployment.environment` sets the environment that the application runs in. This will be diplayed as the **Hosted In** environment of your application in Application Signals dashboards. If you don't specify this, the default of `generic:default` is used. This attribute key is used only by Application Signals, and is converted into X-Ray

Environment variable	Setting to enable Application Signals
	trace annotations and CloudWatch metric dimensions. (Optional) To enable log correlation for Application Signals, set an additional environment variable `aws.log.group.names` to be the log group name for your application log. By doing so, the traces and metrics from your application can be correlated with the relevant log entries from the log group. For this variable, replace *$YOUR_APPLICATION_LOG_GROUP* with the log group names for your application. If you have multiple log groups, you can use an ampersand (&) to separate them as in this example: `aws.log.group.names=log-group-1&log-group-2` . To enable metric to log correlation, setting this current environmental variable is enough. For more information, see [Enable metric to log correlation](). To enable trace to log correlation, you'll also need to change the logging configuration in your application. For more information, see [Enable trace to log correlation]().
`OTEL_AWS_APPLICATION_SIGNALS_ENABLED`	Set to `true` to have your container start sending X-Ray traces and CloudWatch metrics to Application Signals.
`OTEL_METRICS_EXPORTER`	Set to none to disable other metrics exporters.
`OTEL_LOGS_EXPORTER`	Set to none to disable other logs exporters.

Environment variable	Setting to enable Application Signals
`OTEL_EXPORTER_OTLP_PROTOCOL`	Set to `http/protobuf` to send metrics and traces to Application Signals using OTLP/HTTP and protobuf.
`OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT`	Sends metrics to the CloudWatch daemon container. • For applications running in `host` mode, set this to `http://localhost:4316/v1/metrics` . • For applications running in `bridge` mode or `awsvpc` mode, set this to http://*`CW_CONTAINER_IP`* :4316/v1/metrics, where *`CW_CONTAINER_IP`* is the private IP address of the EC2 container instance. You can retrieve this address from the Instance Metadata Service (IMDS).
`OTEL_EXPORTER_OTLP_TRACES_ENDPOINT`	Sends traces to the CloudWatch daemon container. • For applications running in `host` mode, set this to `http://localhost:4316/v1/traces` . • For applications running in `bridge` mode or `awsvpc` mode, set this to http://*`CW_CONTAINER_IP`* :4316/v1/traces, where *`CW_CONTAINER_IP`* is the private IP address of the EC2 container instance. You can retrieve this address from the Instance Metadata Service (IMDS).

Environment variable	Setting to enable Application Signals
`OTEL_TRACES_SAMPLER`	Set this to `xray` to set X-Ray as the traces sampler.
`OTEL_TRACES_SAMPLER_ARG`	Sets the traces sampler endpoint. • For applications running in `host` mode, set this to `http://localhost:2000` . • For applications running in `bridge` mode or `awsvpc` mode, set this to `http://CW_CONTAINER_IP :2000`, where *CW_CONTAINER_IP* is the private IP address of the EC2 container instance. You can retrieve this address from the Instance Metadata Service (IMDS).
`OTEL_PROPAGATORS`	Set `xray` as one of the propagators.
`NODE_OPTIONS`	Set to `--require AWS_ADOT_NODE_INSTRUMENTATION_PATH` . Replace *AWS_ADOT_NODE_INSTRUMENTATION_PATH* with the path where the Amazon Distro for OpenTelemetry Node.js auto-instrumentation is stored. For example, `/otel-auto-instrumentation-node/autoinstrumentation.js`

5. Mount the volume `opentelemetry-auto-instrumentation-node` that you defined in step 1 of this procedure. If you don't need to enable log correlation with metrics and traces, use the following example for a Node.js application. If you want to enable log correlation, see the next step instead.

   For your Application Container, add a dependency on the `init` container to make sure that container finishes before your application container starts.

```
{
  "name": "my-app",
   ...
  "environment": [
    {
      "name": "OTEL_RESOURCE_ATTRIBUTES",
      "value": "service.name=$SVC_NAME"
    },
    {
      "name": "OTEL_LOGS_EXPORTER",
      "value": "none"
    },
    {
      "name": "OTEL_METRICS_EXPORTER",
      "value": "none"
    },
    {
      "name": "OTEL_EXPORTER_OTLP_PROTOCOL",
      "value": "http/protobuf"
    },
    {
      "name": "OTEL_AWS_APPLICATION_SIGNALS_ENABLED",
      "value": "true"
    },
    {
      "name": "OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT",
      "value": "http://CW_CONTAINER_IP:4316/v1/metrics"
    },
    {
      "name": "OTEL_EXPORTER_OTLP_TRACES_ENDPOINT",
      "value": "http://CW_CONTAINER_IP:4316/v1/traces"
    },
    {
      "name": "OTEL_TRACES_SAMPLER",
      "value": "xray"
    },
    {
      "name": "OTEL_TRACES_SAMPLER_ARG",
      "value": "endpoint=http://CW_CONTAINER_IP:2000"
    },
    {
      "name": "NODE_OPTIONS",
```

```
      "value": "--require /otel-auto-instrumentation-node/
autoinstrumentation.js"
    }
    ],
  "mountPoints": [
    {
      "sourceVolume": "opentelemetry-auto-instrumentation-node",
      "containerPath": "/otel-auto-instrumentation-node",
      "readOnly": false
    }
  ],
  "dependsOn": [
    {
      "containerName": "init",
      "condition": "SUCCESS"
    }
  ]
}
```

6. (Optional) To enable log correlation, do the following before you mount the volume. In `OTEL_RESOURCE_ATTRIBUTES`, set an additional environment variable `aws.log.group.names` for the log groups of your application. By doing so, the traces and metrics from your application can be correlated with the relevant log entries from these log groups. For this variable, replace *$YOUR_APPLICATION_LOG_GROUP* with the log group names for your application. If you have multiple log groups, you can use an ampersand (&) to separate them as in this example: `aws.log.group.names=log-group-1&log-group-2`. To enable metric to log correlation, setting this current environmental variable is enough. For more information, see Enable metric to log correlation. To enable trace to log correlation, you'll also need to change the logging configuration in your application. For more information, see Enable trace to log correlation.

   The following is an example. Use this example to enable log correlation when you mount the volume `opentelemetry-auto-instrumentation` that you defined in step 1 of this procedure.

```
{
  "name": "my-app",
   ...
  "environment": [
    {
      "name": "OTEL_RESOURCE_ATTRIBUTES",
```

```
        "value":
 "aws.log.group.names=$YOUR_APPLICATION_LOG_GROUP,service.name=$SVC_NAME"
      },
      {
        "name": "OTEL_LOGS_EXPORTER",
        "value": "none"
      },
      {
        "name": "OTEL_METRICS_EXPORTER",
        "value": "none"
      },
      {
        "name": "OTEL_EXPORTER_OTLP_PROTOCOL",
        "value": "http/protobuf"
      },
      {
        "name": "OTEL_AWS_APPLICATION_SIGNALS_ENABLED",
        "value": "true"
      },
      {
        "name": "OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT",
        "value": "http://CW_CONTAINER_IP:4316/v1/metrics"
      },
      {
        "name": "OTEL_EXPORTER_OTLP_TRACES_ENDPOINT",
        "value": "http://CW_CONTAINER_IP:4316/v1/traces"
      },
      {
        "name": "OTEL_TRACES_SAMPLER",
        "value": "xray"
      },
      {
        "name": "OTEL_TRACES_SAMPLER_ARG",
        "value": "endpoint=http://CW_CONTAINER_IP:2000"
      },
      {
        "name": "NODE_OPTIONS",
        "value": "--require /otel-auto-instrumentation-node/
autoinstrumentation.js"
      }
      ],
    "mountPoints": [
      {
        "sourceVolume": "opentelemetry-auto-instrumentation-node",
```

```
        "containerPath": "/otel-auto-instrumentation-node",
        "readOnly": false
      }
    ],
    "dependsOn": [
      {
        "containerName": "init",
        "condition": "SUCCESS"
      }
    ]
}
```

Setting up a Node.js application with the ESM module format

We provide limited support for Node.js applications with the ESM module format. For details, see the section called “Known limitations about Node.js with ESM”.

For the ESM module format, using the `init` container to inject the Node.js instrumentation SDK doesn’t apply. To enable Application Signals for Node.js with ESM, skip steps 1 and 2 in of the previous procedure, and do the following instead.

To enable Application Signals for a Node.js application with ESM

1. Install the relevant dependencies to your Node.js application for autoinstrumentation:

```
npm install @aws/aws-distro-opentelemetry-node-autoinstrumentation
npm install @opentelemetry/instrumentation@0.54.0
```

2. In steps 4 and 5 in the previous procedure, remove the mounting of the volume `opentelemetry-auto-instrumentation-node`:

```
"mountPoints": [
    {
        "sourceVolume": "opentelemetry-auto-instrumentation-node",
        "containerPath": "/otel-auto-instrumentation-node",
        "readOnly": false
    }
 ]
```

Replace the node options with the following.

```
{
    "name": "NODE_OPTIONS",
    "value": "--import @aws/aws-distro-opentelemetry-node-autoinstrumentation/
register --experimental-loader=@opentelemetry/instrumentation/hook.mjs"
}
```

Step 6: Deploy your application

Create a new revision of your task definition and deploy it to your application cluster. You should see two containers in the newly created task:

- `init`– A required container for initializing Application Signals
- *`my-app`*– This is the example application container in our documentation. In your actual workloads, this specific container might not exist or might be replaced with your own service containers.

(Optional) Step 7: Monitor your application health

Once you have enabled your applications on Amazon ECS, you can monitor your application health. For more information, see Monitor the operational health of your applications with Application Signals.

Enable Application Signals on Amazon ECS using Amazon CDK

To enable Application Signals on Amazon ECS using Amazon CDK, do the following.

1. Enable Application Signals for your applications – If you haven't enabled Application Signals in this account yet, you must grant Application Signals the permissions it needs to discover your services.

```
import { aws_applicationsignals as applicationsignals } from 'aws-cdk-lib';

const cfnDiscovery = new applicationsignals.CfnDiscovery(this,
  'ApplicationSignalsServiceRole', { }
);
```

The Discovery CloudFormation resource grants Application Signals the following permissions:

- `xray:GetServiceGraph`
- `logs:StartQuery`
- `logs:GetQueryResults`
- `cloudwatch:GetMetricData`
- `cloudwatch:ListMetrics`
- `tag:GetResources`

For more information about this role, see Service-linked role permissions for CloudWatch Application Signals.

2. Instrument your application with the AWS::ApplicationSignals Construct Library in the Amazon CDK. The code snippets in this document are provided in *TypeScript*. For other language-specific alternatives, see Supported programming languages for the Amazon CDK.

   - **Enable Application Signals on Amazon ECS with sidecar mode**

     a. Configure `instrumentation` to instrument the application with the Amazon Distro for OpenTelemetry (ADOT) SDK Agent. The following is an example of instrumenting a Java application. See InstrumentationVersion for all supported language versions.

     b. Specify `cloudWatchAgentSidecar` to configure the CloudWatch Agent as a sidecar container.

```
import { Construct } from 'constructs';
import * as appsignals from '@aws-cdk/aws-applicationsignals-alpha';
import * as cdk from 'aws-cdk-lib';
import * as ec2 from 'aws-cdk-lib/aws-ec2';
import * as ecs from 'aws-cdk-lib/aws-ecs';

class MyStack extends cdk.Stack {
  public constructor(scope?: Construct, id?: string, props: cdk.StackProps
 = {}) {
    super();
    const vpc = new ec2.Vpc(this, 'TestVpc', {});
    const cluster = new ecs.Cluster(this, 'TestCluster', { vpc });

    const fargateTaskDefinition = new ecs.FargateTaskDefinition(this,
 'SampleAppTaskDefinition', {
```

```
      cpu: 2048,
      memoryLimitMiB: 4096,
    });

    fargateTaskDefinition.addContainer('app', {
      image: ecs.ContainerImage.fromRegistry('test/sample-app'),
    });

    new appsignals.ApplicationSignalsIntegration(this,
 'ApplicationSignalsIntegration', {
      taskDefinition: fargateTaskDefinition,
      instrumentation: {
        sdkVersion: appsignals.JavaInstrumentationVersion.V2_10_0,
      },
      serviceName: 'sample-app',
      cloudWatchAgentSidecar: {
        containerName: 'ecs-cwagent',
        enableLogging: true,
        cpu: 256,
        memoryLimitMiB: 512,
      }
    });

    new ecs.FargateService(this, 'MySampleApp', {
      cluster: cluster,
      taskDefinition: fargateTaskDefinition,
      desiredCount: 1,
    });
  }
}
```

- **Enable Application Signals on Amazon ECS with daemon mode**

  Note

  The daemon deployment strategy is not supported on Amazon ECS Fargate and is only supported on Amazon ECS on Amazon EC2.

  a. Run CloudWatch Agent as a daemon service with `HOST` network mode.

  b. Configure `instrumentation` to instrument the application with the ADOT Python Agent.

```
import { Construct } from 'constructs';
import * as appsignals from '@aws-cdk/aws-applicationsignals-alpha';
import * as cdk from 'aws-cdk-lib';
import * as ec2 from 'aws-cdk-lib/aws-ec2';
import * as ecs from 'aws-cdk-lib/aws-ecs';

class MyStack extends cdk.Stack {
  public constructor(scope?: Construct, id?: string, props: cdk.StackProps
 = {}) {
    super(scope, id, props);

    const vpc = new ec2.Vpc(this, 'TestVpc', {});
    const cluster = new ecs.Cluster(this, 'TestCluster', { vpc });

    // Define Task Definition for CloudWatch agent (Daemon)
    const cwAgentTaskDefinition = new ecs.Ec2TaskDefinition(this,
 'CloudWatchAgentTaskDefinition', {
      networkMode: ecs.NetworkMode.HOST,
    });

    new appsignals.CloudWatchAgentIntegration(this,
 'CloudWatchAgentIntegration', {
      taskDefinition: cwAgentTaskDefinition,
      containerName: 'ecs-cwagent',
      enableLogging: false,
      cpu: 128,
      memoryLimitMiB: 64,
      portMappings: [
        {
          containerPort: 4316,
          hostPort: 4316,
        },
        {
          containerPort: 2000,
          hostPort: 2000,
        },
      ],
    });

    // Create the CloudWatch Agent daemon service
    new ecs.Ec2Service(this, 'CloudWatchAgentDaemon', {
      cluster,
      taskDefinition: cwAgentTaskDefinition,
```

```
      daemon: true,  // Runs one container per EC2 instance
    });

    // Define Task Definition for user application
    const sampleAppTaskDefinition = new ecs.Ec2TaskDefinition(this,
 'SampleAppTaskDefinition', {
      networkMode: ecs.NetworkMode.HOST,
    });

    sampleAppTaskDefinition.addContainer('app', {
      image: ecs.ContainerImage.fromRegistry('test/sample-app'),
      cpu: 0,
      memoryLimitMiB: 512,
    });

    // No CloudWatch Agent sidecar is needed as application container
 communicates to CloudWatch Agent daemon through host network
    new appsignals.ApplicationSignalsIntegration(this,
 'ApplicationSignalsIntegration', {
      taskDefinition: sampleAppTaskDefinition,
      instrumentation: {
        sdkVersion: appsignals.PythonInstrumentationVersion.V0_8_0
      },
      serviceName: 'sample-app'
    });

    new ecs.Ec2Service(this, 'MySampleApp', {
      cluster,
      taskDefinition: sampleAppTaskDefinition,
      desiredCount: 1,
    });
  }
}
```

- **Enable Application Signals on Amazon ECS with replica mode**

> **Note**
>
> Running CloudWatch Agent service using replica mode requires specific security group configurations to enable communication with other services. For Application Signals functionality, configure the security group with the minimum inbound

rules: Port 2000 (HTTP) and Port 4316 (HTTP). This configuration ensures proper connectivity between the CloudWatch Agent and dependent services.

a. Run CloudWatch Agent as a replica service with service connect.

b. Configure `instrumentation` to instrument the application with the ADOT Python Agent.

c. Override environment variables by configuring `overrideEnvironments` to use service connect endpoints to communicate to the CloudWatch agent server.

```
import { Construct } from 'constructs';
import * as appsignals from '@aws-cdk/aws-applicationsignals-alpha';
import * as cdk from 'aws-cdk-lib';
import * as ec2 from 'aws-cdk-lib/aws-ec2';
import * as ecs from 'aws-cdk-lib/aws-ecs';
import { PrivateDnsNamespace } from 'aws-cdk-lib/aws-servicediscovery';

class MyStack extends cdk.Stack {
  public constructor(scope?: Construct, id?: string, props: cdk.StackProps
 = {}) {
    super(scope, id, props);

    const vpc = new ec2.Vpc(this, 'TestVpc', {});
    const cluster = new ecs.Cluster(this, 'TestCluster', { vpc });
    const dnsNamespace = new PrivateDnsNamespace(this, 'Namespace', {
      vpc,
      name: 'local',
    });
    const securityGroup = new ec2.SecurityGroup(this, 'ECSSG', { vpc });
    securityGroup.addIngressRule(securityGroup, ec2.Port.tcpRange(0,
 65535));

    // Define Task Definition for CloudWatch agent (Replica)
    const cwAgentTaskDefinition = new ecs.FargateTaskDefinition(this,
 'CloudWatchAgentTaskDefinition', {});

    new appsignals.CloudWatchAgentIntegration(this,
 'CloudWatchAgentIntegration', {
      taskDefinition: cwAgentTaskDefinition,
      containerName: 'ecs-cwagent',
      enableLogging: false,
```

```
      cpu: 128,
      memoryLimitMiB: 64,
      portMappings: [
        {
          name: 'cwagent-4316',
          containerPort: 4316,
          hostPort: 4316,
        },
        {
          name: 'cwagent-2000',
          containerPort: 2000,
          hostPort: 2000,
        },
      ],
    });

    // Create the CloudWatch Agent replica service with service connect
    new ecs.FargateService(this, 'CloudWatchAgentService', {
      cluster: cluster,
      taskDefinition: cwAgentTaskDefinition,
      securityGroups: [securityGroup],
      serviceConnectConfiguration: {
        namespace: dnsNamespace.namespaceArn,
        services: [
          {
            portMappingName: 'cwagent-4316',
            dnsName: 'cwagent-4316-http',
            port: 4316,
          },
          {
            portMappingName: 'cwagent-2000',
            dnsName: 'cwagent-2000-http',
            port: 2000,
          },
        ],
      },
      desiredCount: 1,
    });

    // Define Task Definition for user application
    const sampleAppTaskDefinition = new ecs.FargateTaskDefinition(this,
 'SampleAppTaskDefinition', {});

    sampleAppTaskDefinition.addContainer('app', {
```

```
      image: ecs.ContainerImage.fromRegistry('test/sample-app'),
      cpu: 0,
      memoryLimitMiB: 512,
    });

    // Overwrite environment variables to connect to the CloudWatch Agent
 service just created
    new appsignals.ApplicationSignalsIntegration(this,
 'ApplicationSignalsIntegration', {
      taskDefinition: sampleAppTaskDefinition,
      instrumentation: {
        sdkVersion: appsignals.PythonInstrumentationVersion.V0_8_0,
      },
      serviceName: 'sample-app',
      overrideEnvironments: [
        {
          name:
 appsignals.CommonExporting.OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT,
          value: 'http://cwagent-4316-http:4316/v1/metrics',
        },
        {
          name:
 appsignals.TraceExporting.OTEL_EXPORTER_OTLP_TRACES_ENDPOINT,
          value: 'http://cwagent-4316-http:4316/v1/traces',
        },
        {
          name: appsignals.TraceExporting.OTEL_TRACES_SAMPLER_ARG,
          value: 'endpoint=http://cwagent-2000-http:2000',
        },
      ],
    });

    // Create ECS Service with service connect configuration
    new ecs.FargateService(this, 'MySampleApp', {
      cluster: cluster,
      taskDefinition: sampleAppTaskDefinition,
      serviceConnectConfiguration: {
        namespace: dnsNamespace.namespaceArn,
      },
      desiredCount: 1,
    });
  }
}
```

3. Setting up a Node.js application with the ESM module format. There is limited support for Node.js applications with the ESM module format. For more information, see [Known limitations about Node.js with ESM](#).

   For the ESM module format, enabling Application Signals by using the `init` container to inject the Node.js instrumentation SDK doesn't apply. Skip Step 2 in this procedure, and do the following instead.

   - Install the relevant dependencies to your Node.js application for autoinstrumentation.

     ```
     npm install @aws/aws-distro-opentelemetry-node-autoinstrumentation
     npm install @opentelemetry/instrumentation@0.54.
     ```

   - Update TaskDefinition.

     a. Add additional configuration to your application container.

     b. Configure `NODE_OPTIONS`.

     c. (Optional) Add CloudWatch Agent if you choose sidecar mode.

     ```
     import { Construct } from 'constructs';
     import * as appsignals from '@aws-cdk/aws-applicationsignals-alpha';
     import * as ecs from 'aws-cdk-lib/aws-ecs';

     class MyStack extends cdk.Stack {
       public constructor(scope?: Construct, id?: string, props: cdk.StackProps
      = {}) {
         super(scope, id, props);
         const fargateTaskDefinition = new ecs.FargateTaskDefinition(stack,
      'TestTaskDefinition', {
           cpu: 256,
           memoryLimitMiB: 512,
         });
         const appContainer = fargateTaskDefinition.addContainer('app', {
           image: ecs.ContainerImage.fromRegistry('docker/cdk-test'),
         });

         const volumeName = 'opentelemetry-auto-instrumentation'
         fargateTaskDefinition.addVolume({name: volumeName});

         // Inject additional configurations
         const injector = new appsignals.NodeInjector(volumeName,
      appsignals.NodeInstrumentationVersion.V0_5_0);
     ```

```
    injector.renderDefaultContainer(fargateTaskDefinition);
    // Configure NODE_OPTIONS
    appContainer.addEnvironment('NODE_OPTIONS', '--import @aws/aws-
distro-opentelemetry-node-autoinstrumentation/register --experimental-
loader=@opentelemetry/instrumentation/hook.mjs')

    // Optional: add CloudWatch agent
    const cwAgent = new appsignals.CloudWatchAgentIntegration(stack,
 'AddCloudWatchAgent', {
      containerName: 'ecs-cwagent',
      taskDefinition: fargateTaskDefinition,
      memoryReservationMiB: 50,
    });
    appContainer.addContainerDependencies({
      container: cwAgent.agentContainer,
      condition: ecs.ContainerDependencyCondition.START,
    });
}
```

4. Deploy the updated stack – Run the `cdk synth` command in your application's main directory. To deploy the service in your Amazon account, run the `cdk deploy` command in your application's main directory.

   If you used the sidecar strategy, you'll see one service created:

   - *`APPLICATION_SERVICE`* is the service of your application. It includes the three following containers:
     - `init`– A required container for initializing Application Signals.
     - `ecs-cwagent`– A container running the CloudWatch agent
     - *`my-app`*– This is the example application container in our documentation. In your actual workloads, this specific container might not exist or might be replaced with your own service containers.

   If you used the daemon strategy, you'll see two services created:

   - *`CloudWatchAgentDaemon`* is the CloudWatch agent daemon service.
   - *`APPLICATION_SERVICE`* is the service of your application. It includes the two following containers:
     - `init`– A required container for initializing Application Signals.

    - *my-app*– This is the example application container in our documentation. In your actual workloads, this specific container might not exist or might be replaced with your own service containers.

If you used the replica strategy, you'll see two services created:

- *CloudWatchAgentService* is the CloudWatch agent replica service.
- *APPLICATION_SERVICE* is the service of your application. It includes the two following containers:
    - `init`– A required container for initializing Application Signals.
    - *my-app*– This is the example application container in our documentation. In your actual workloads, this specific container might not exist or might be replaced with your own service containers.

Enable your applications on Kubernetes

Enable CloudWatch Application Signals on Kubernetes by using the custom setup steps described in this section.

For applications running on Kubernetes, you install and configure the CloudWatch agent and Amazon Distro for OpenTelemetry yourself. On these architectures enabled with a custom Application Signals setup, Application Signals doesn't autodiscover the names of your services or the hosts or clusters they run on. You must specify these names during the custom setup, and the names that you specify are what is displayed on Application Signals dashboards.

Requirements

- You have admininstrator permission on the Kubernetes cluster where you are enabling Application Signals.
- You must have the Amazon CLI installed on the environment where your Kubernetes cluster is running. For more information about installing the Amazon CLI, see [Install or update the latest version of the Amazon CLI].
- You have kubectl and helm installed on your local terminal. For more information, see the [kubectl] and [Helm] documentation.

Step 1: Enable Application Signals in your account

You must first enable Application Signals in your account. If you haven't, see Enable Application Signals in your account.

Step 2: Install the CloudWatch agent operator in your cluster

Installing the CloudWatch agent operator installs the operator, the CloudWatch agent, and other auto-instrumentation into your cluster. To do so, enter the following command. Replace *$REGION* with your Amazon Region. Replace *$YOUR_CLUSTER_NAME* with the name that you want to appear for your cluster in Application Signals dashboards.

```
helm repo add aws-observability https://aws-observability.github.io/helm-charts
helm install amazon-cloudwatch-operator aws-observability/amazon-cloudwatch-observability \
--namespace amazon-cloudwatch --create-namespace \
--set region=$REGION \
--set clusterName=$YOUR_CLUSTER_NAME
```

For more information, see amazon-cloudwatch-observability on GitHub.

Step 3: Set up Amazon credentials for your Kubernetes clusters

> **Important**
>
> If your Kubernetes cluster is hosted on Amazon EC2, you can skip this section and proceed to Step 4: Add annotations.

If your Kubernetes cluster is hosted on-premises, you must use the instructions in this section to add Amazon credentials to your Kubernetes environment.

To set up permissions for an on-premises Kubernetes cluster

1. Create the IAM user to be used to provide permissions to your on-premises host:

   a. Open the IAM console at https://console.amazonaws.cn/iam/.

   b. Choose **Users**, **Create User**.

   c. In **User details**, for **User name**, enter a name for the new IAM user. This is the sign-in name for Amazon that will be used to authenticate your host. Then choose **Next**

d. On the **Set permissions** page, under **Permissions options**, select **Attach policies directly**.

e. From the **Permissions policies** list, select the **CloudWatchAgentServerPolicy** policy to add to your user. Then choose **Next**.

f. On the **Review and create** page, ensure that you are satisfied with the user name and that the **CloudWatchAgentServerPolicy** policy is in the **Permissions summary**.

g. Choose **Create user**

2. Create and retrieve your Amazon access key and secret key:

   a. In the navigation pane in the IAM console, choose **Users** and then select the user name of the user that you created in the previous step.

   b. On the user's page, choose the **Security credentials** tab. Then, in the **Access keys** section, choose **Create access key**.

   c. For **Create access key Step 1**, choose **Command Line Interface (CLI)**.

   d. For **Create access key Step 2**, optionally enter a tag and then choose **Next**.

   e. For **Create access key Step 3**, select **Download .csv file** to save a .csv file with your IAM user's access key and secret access key. You need this information for the next steps.

   f. Choose **Done**.

3. Configure your Amazon credentials in your on-premises host by entering the following command. Replace *ACCESS_KEY_ID* and *SECRET_ACCESS_ID* with your newly generated access key and secret access key from the .csv file that you downloaded in the previous step. By default, the credential file is saved in `/home/user/.aws/credentials.`

```
$ aws configure --profile AmazonCloudWatchAgent
AWS Access Key ID [None]: ACCESS_KEY_ID
AWS Secret Access Key [None]: SECRET_ACCESS_ID
Default region name [None]: MY_REGION
Default output format [None]: json
```

4. Edit the custom resource that the CloudWatch agent installed using the Helm chart to add the newly created Amazon credentials secret.

```
kubectl edit amazoncloudwatchagent cloudwatch-agent -n amazon-cloudwatch
```

5. While your file editor is open mount the Amazon credentials into the CloudWatch agent container by adding the following configuration to the top of the deployment. Replace the

path /home/*user*/.aws/credentials with the location of your local Amazon credentials file.

```
apiVersion: cloudwatch.aws.amazon.com/v1alpha1
kind: AmazonCloudWatchAgent
metadata:
  name: cloudwatch-agent
  namespace: amazon-cloudwatch
spec:
  volumeMounts:
  - mountPath: /rootfs
    volumeMounts:
    - name: aws-credentials
      mountPath: /root/.aws
      readOnly: true
    volumes:
    - hostPath:
      path: /home/user/.aws/credentials
    name: aws-credentials
---
```

Step 4: Add annotations

> **Note**
>
> If you are enabling Application Signals for a Node.js application with ESM, skip the steps in this section and see the section called "Setting up a Node.js application with the ESM module format" instead.

The next step is to instrument your application for CloudWatch Application Signals by adding a language-specific annotation to your Kubernetes workload or namespace. This annotation auto-instruments your application to send metrics, traces, and logs to Application Signals.

To add the annotations for Application Signals

1. You have two options for the annotation:

   - **Annotate Workload** auto-instruments a single workload in a cluster.
   - **Annotate Namespace** auto-instruments all workloads deployed in the selected namespace.

Choose one of those options, and follow the appropriate steps.

2. To annotate a single workload, enter one of the following commands. Replace *$WORKLOAD_TYPE* and *$WORKLOAD_NAME* with the values for your workload.

   - For Java workloads:

     ```
     kubectl patch $WORKLOAD_TYPE $WORKLOAD_NAME -p '{"spec": {"template":
      {"metadata": {"annotations": {"instrumentation.opentelemetry.io/inject-java":
      "true"}}}}}'
     ```

   - For Python workloads:

     ```
     kubectl patch $WORKLOAD_TYPE $WORKLOAD_NAME -p '{"spec": {"template":
      {"metadata": {"annotations": {"instrumentation.opentelemetry.io/inject-python":
      "true"}}}}}'
     ```

     For Python applications, there are additional required configurations. For more information, see Python application doesn't start after Application Signals is enabled.

   - For .NET workloads:

     ```
     kubectl patch $WORKLOAD_TYPE $WORKLOAD_NAME -p '{"spec": {"template":
      {"metadata": {"annotations": {"instrumentation.opentelemetry.io/inject-dotnet":
      "true"}}}}}'
     ```

     Note

     To enable Application Signals for a .NET workload on Alpine Linux (`linux-musl-x64`) based images, add the following additional annotation.

     ```
     instrumentation.opentelemetry.io/otel-dotnet-auto-runtime: "linux-musl-
     x64"
     ```

   - For Node.js workloads:

     ```
     kubectl patch $WORKLOAD_TYPE $WORKLOAD_NAME -p '{"spec": {"template":
      {"metadata": {"annotations": {"instrumentation.opentelemetry.io/inject-nodejs":
      "true"}}}}}'
     ```

3. To annotate all workloads in a namespace, enter enter one of the following commands. Replace *$NAMESPACE* with the name of your namespace.

   If the namespace includes Java, Python, and .NET workloads, add all annotations to the namespace.

   - For Java workloads in the namespace:

     ```
     kubectl annotate ns $NAMESPACE instrumentation.opentelemetry.io/inject-java=true
     ```

   - For Python workloads in the namespace:

     ```
     kubectl annotate ns $NAMESPACE instrumentation.opentelemetry.io/inject-
     python=true
     ```

     For Python applications, there are additional required configurations. For more information, see Python application doesn't start after Application Signals is enabled.

   - For .NET workloads in the namespace:

     ```
     kubectl annotate ns $NAMESPACE instrumentation.opentelemetry.io/inject-
     dotnet=true
     ```

   - For Node.js workloads in the namespace:

     ```
     kubectl annotate ns $NAMESPACE instrumentation.opentelemetry.io/inject-
     nodejs=true
     ```

   After adding the annotations, restart all pods in the namespace by entering the following command:

   ```
   kubectl rollout restart
   ```

4. When the previous steps are completed, in the CloudWatch console, choose **Application Signals**, **Services**. This opens the dashboards where you can see the data that Application Signals collects. It might take a few minutes for data to appear.

   For more information about the **Services** view, see Monitor the operational health of your applications with Application Signals.

Setting up a Node.js application with the ESM module format

We provide limited support for Node.js applications with the ESM module format. For details, see the section called "Known limitations about Node.js with ESM".

For the ESM module format, enabling Application Signals by annotating the manifest file doesn't work. Skip the previous procedure and do the following instead:

To enable Application Signals for a Node.js application with ESM

1. Install the relevant dependencies to your Node.js application for autoinstrumentation:

```
npm install @aws/aws-distro-opentelemetry-node-autoinstrumentation
npm install @opentelemetry/instrumentation@0.54.0
```

2. Add the following environmental variables to the Dockerfile for your application and build the image.

```

...
ENV OTEL_AWS_APPLICATION_SIGNALS_ENABLED=true
ENV OTEL_TRACES_SAMPLER_ARG='endpoint=http://cloudwatch-agent.amazon-
cloudwatch:2000'
ENV OTEL_TRACES_SAMPLER='xray'
ENV OTEL_EXPORTER_OTLP_PROTOCOL='http/protobuf'
ENV OTEL_EXPORTER_OTLP_TRACES_ENDPOINT='http://cloudwatch-agent.amazon-
cloudwatch:4316/v1/traces'
ENV OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT='http://cloudwatch-agent.amazon-
cloudwatch:4316/v1/metrics'
ENV OTEL_METRICS_EXPORTER='none'
ENV OTEL_LOGS_EXPORTER='none'
ENV NODE_OPTIONS='--import @aws/aws-distro-opentelemetry-node-autoinstrumentation/
register --experimental-loader=@opentelemetry/instrumentation/hook.mjs'
ENV OTEL_SERVICE_NAME='YOUR_SERVICE_NAME' #replace with a proper service name
ENV OTEL_PROPAGATORS='tracecontext,baggage,b3,xray'
...

# command to start the application
# for example
# CMD ["node", "index.mjs"]
```

3. Add the environmental variables `OTEL_RESOURCE_ATTRIBUTES_POD_NAME`, `OTEL_RESOURCE_ATTRIBUTES_NODE_NAME`,

OTEL_RESOURCE_ATTRIBUTES_DEPLOYMENT_NAME, POD_NAMESPACE and OTEL_RESOURCE_ATTRIBUTES to the deployment yaml file for the application. For example:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nodejs-app
  labels:
    app: nodejs-app
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nodejs-app
  template:
    metadata:
      labels:
        app: nodejs-app
      # annotations:
      # make sure this annotation doesn't exit
      #   instrumentation.opentelemetry.io/inject-nodejs: 'true'
    spec:
      containers:
      - name: nodejs-app
        image:your-nodejs-application-image #replace it with a proper image uri
        imagePullPolicy: Always
        ports:
        - containerPort: 8000
        env:
          - name: OTEL_RESOURCE_ATTRIBUTES_POD_NAME
            valueFrom:
              fieldRef:
                fieldPath: metadata.name
          - name: OTEL_RESOURCE_ATTRIBUTES_NODE_NAME
            valueFrom:
              fieldRef:
                fieldPath: spec.nodeName
          - name: OTEL_RESOURCE_ATTRIBUTES_DEPLOYMENT_NAME
            valueFrom:
              fieldRef:
                fieldPath: metadata.labels['app'] # Assuming 'app' label is set to
 the deployment name
          - name: POD_NAMESPACE
```

```
            valueFrom:
              fieldRef:
                fieldPath: metadata.namespace
          - name: OTEL_RESOURCE_ATTRIBUTES
            value: "k8s.deployment.name=
$(OTEL_RESOURCE_ATTRIBUTES_DEPLOYMENT_NAME),k8s.namespace.name=
$(POD_NAMESPACE),k8s.node.name=$(OTEL_RESOURCE_ATTRIBUTES_NODE_NAME),k8s.pod.name=
$(OTEL_RESOURCE_ATTRIBUTES_POD_NAME)"
```

4. Deploy the Node.js application to the Kubernetes cluster.

(Optional) Step 5: Monitor your application health

Once you have enabled your applications on Kubernetes, you can monitor your application health. For more information, see Monitor the operational health of your applications with Application Signals.

Enable your applications on Lambda

You can enable Application Signals for your Lambda functions. Application Signals automatically instruments your Lambda functions using enhanced Amazon Distro for OpenTelemetry (ADOT) libraries, provided through a Lambda layer. This Amazon Lambda Layer for OpenTelemetry packages and deploys the libraries that are required for auto-instrumentation for Application Signals.

In addition to supporting Application Signals, this Lambda layer is also a component of Lambda OpenTelemetry support and provides tracing functionality.

Topics

- [Amazon Lambda Layer for OpenTelemetry ARNs]
- [Deploy Lambda functions using Amazon ECR container]

Getting started

There are three methods for enabling Application Signals for your Lambda functions.

After you enable Application Signals for a Lambda function, it takes a few minutes for telemetry from that function to appear in the Application Signals console.

- Use the CloudWatch Application Signals console
- Use the Lambda console
- Manually add the Amazon Lambda Layer for OpenTelemetry to your Lambda function runtime.

Each of these methods adds the Amazon Lambda Layer for OpenTelemetry to your function.

Use the CloudWatch Application Signals console

Use these steps to use the Application Signals console to enable Application Signals for a Lambda function.

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Application Signals**, **Services**.
3. In the **Services** list area, choose **Enable Application Signals**.
4. Choose the **Lambda** tile.
5. Select each function that you want to enable for Application Signals, and then choose **Done**.

Use the Lambda console

Use these steps to use the Lambda console to enable Application Signals for a Lambda function.

1. Open the Amazon Lambda console at https://console.amazonaws.cn/lambda/.
2. In the navigation pane, choose **Functions** and then choose the name of the function that you want to enable.
3. Choose the **Configuration** tab, and then choose **Monitoring and operations tools**.

4. Choose **Edit**.
5. In the **CloudWatch Application Signals and X-Ray** section, select both **Automatically collect application traces and standard application metrics with Application Signals** and **Automatically collect Lambda service traces for end to end visibility with X-Ray.**.
6. Choose **Save**.

Enable Application Signals on Lambda using Amazon CDK

If you haven't enabled Application Signals in this account yet, you must grant Application Signals the permissions it needs to discover your services. For more information, see Enable Application Signals in your account.

1. Enable Application Signals for your applications

```
import { aws_applicationsignals as applicationsignals } from 'aws-cdk-lib';

const cfnDiscovery = new applicationsignals.CfnDiscovery(this,
  'ApplicationSignalsServiceRole', { }
);
```

   The Discovery CloudFormation resource grants Application Signals the following permissions:

   - `xray:GetServiceGraph`
   - `logs:StartQuery`
   - `logs:GetQueryResults`
   - `cloudwatch:GetMetricData`
   - `cloudwatch:ListMetrics`
   - `tag:GetResources`

   For more information about this role, see Service-linked role permissions for CloudWatch Application Signals.

2. Add the IAM policy `CloudWatchLambdaApplicationSignalsExecutionRolePolicy` to the lambda function.

```
const fn = new Function(this, 'DemoFunction', {
    code: Code.fromAsset('$YOUR_LAMBDA.zip'),
```

```
    runtime: Runtime.PYTHON_3_12,
    handler: '$YOUR_HANDLER'
})

fn.role?.addManagedPolicy(ManagedPolicy.fromAwsManagedPolicyName('CloudWatchLambdaApplicati
```

3. Replace $AWS_LAMBDA_LAYER_FOR_OTEL_ARN with the actual Amazon Lambda Layer for OpenTelemetry ARN in the corresponding region.

```
fn.addLayers(LayerVersion.fromLayerVersionArn(
    this, 'AwsLambdaLayerForOtel',
    '$AWS_LAMBDA_LAYER_FOR_OTEL_ARN'
))
fn.addEnvironment("AWS_LAMBDA_EXEC_WRAPPER", "/opt/otel-instrument");
```

(Optional) Monitor your application health

Once you have enabled your applications on Lambda, you can monitor your application health. For more information, see Monitor the operational health of your applications with Application Signals.

Manually enable Application Signals.

Use these steps to manually enable Application Signals for a Lambda function.

1. Add the Amazon Lambda Layer for OpenTelemetry to your Lambda runtime. To find the layer ARN, see Amazon Lambda Layer for OpenTelemetry ARNs.
2. Add the environment variable `AWS_LAMBDA_EXEC_WRAPPER=/opt/otel-instrument`

   Add the environment variable `LAMBDA_APPLICATION_SIGNALS_REMOTE_ENVIRONMENT` to configure custom Lambda environments. By default, lambda environments are configured to `lambda:default`.
3. Attach the Amazon managed IAM policy **CloudWatchLambdaApplicationSignalsExecutionRolePolicy** to the Lambda execution role.
4. (Optional) We recommend that you enable Lambda active tracing to get a better tracing experience. For more information, see Visualize Lambda function invocations using Amazon X-Ray.

Manually disable Application Signals

To manually disable Application Signals for a Lambda function, remove the Amazon Lambda Layer for OpenTelemetry from your Lambda runtime, and remove the `AWS_LAMBDA_EXEC_WRAPPER=/opt/otel-instrument` environment variable.

Configuring Application Signals

You can use this section to configure Application Signals in Lambda.

Grouping multiple Lambda functions into one service

Environment variable `OTEL_SERVICE_NAME` sets the name of the service. This will be displayed as the service name for your application in Application Signals dashboards. You can assign the same service name to multiple Lambda functions, and they will be merged into a single service in Application Signals. When you don't provide a value for this key, the default Lambda Function name is used.

Sampling

By default, the trace sampling strategy is parent based. You can adjust the sampling strategy by setting environment variables `OTEL_TRACES_SAMPLER`.

For example, set trace sampling rate to 30%.

```
OTEL_TRACES_SAMPLER=traceidratio
OTEL_TRACES_SAMPLER_ARG=0.3
```

For more information , see OpenTelemetry Environment Variable Specification.

Enabling all library instrumentation’s

To reduce Lambda cold starts, by default, only Amazon SDK and HTTP instrumentation’s are enabled for Python, Node, and Java. You can set environment variables to enable instrumentation for other libraries used in your Lambda function.

- Python – `OTEL_PYTHON_DISABLED_INSTRUMENTATIONS=none`
- Node – `OTEL_NODE_DISABLED_INSTRUMENTATIONS=none`
- Java – `OTEL_INSTRUMENTATION_COMMON_DEFAULT_ENABLED=true`

Amazon Lambda Layer for OpenTelemetry ARNs

The following tables list the ARNs to use the Amazon Lambda Layer for OpenTelemetry for each Region where it's supported.

Python

Region	ARN
US East (N. Virginia)	arn:aws:lambda:us-east-1:615299751070:layer:AWSOpenTelemetryDistroPython:12
US East (Ohio)	arn:aws:lambda:us-east-2:615299751070:layer:AWSOpenTelemetryDistroPython:9
US West (N. California)	arn:aws:lambda:us-west-1:615299751070:layer:AWSOpenTelemetryDistroPython:16
US West (Oregon)	arn:aws:lambda:us-west-2:615299751070:layer:AWSOpenTelemetryDistroPython:16
Africa (Cape Town)	arn:aws:lambda:af-south-1:904233096616:layer:AWSOpenTelemetryDistroPython:6
Asia Pacific (Hong Kong)	arn:aws:lambda:ap-east-1:888577020596:layer:AWSOpenTelemetryDistroPython:6
Asia Pacific (Hyderabad)	arn:aws:lambda:ap-south-2:796973505492:layer:AWSOpenTelemetryDistroPython:6
Asia Pacific (Jakarta)	arn:aws:lambda:ap-southeast-3:039612877180:layer:AWSOpenTelemetryDistroPython:6
Asia Pacific (Melbourne)	arn:aws:lambda:ap-southeast-4:713881805771:layer:AWSOpenTelemetryDistroPython:6
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:615299751070:layer:AWSOpenTelemetryDistroPython:9

Region	ARN
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:615299751070:layer:AWSOpenTelemetryDistroPython:8
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:615299751070:layer:AWSOpenTelemetryDistroPython:9
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:615299751070:layer:AWSOpenTelemetryDistroPython:8
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:615299751070:layer:AWSOpenTelemetryDistroPython:9
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:615299751070:layer:AWSOpenTelemetryDistroPython:9
Canada (Central)	arn:aws:lambda:ca-central-1:615299751070:layer:AWSOpenTelemetryDistroPython:9
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:615299751070:layer:AWSOpenTelemetryDistroPython:9
Europe (Ireland)	arn:aws:lambda:eu-west-1:615299751070:layer:AWSOpenTelemetryDistroPython:9
Europe (London)	arn:aws:lambda:eu-west-2:615299751070:layer:AWSOpenTelemetryDistroPython:9
Europe (Milan)	arn:aws:lambda:eu-south-1:257394471194:layer:AWSOpenTelemetryDistroPython:6
Europe (Paris)	arn:aws:lambda:eu-west-3:615299751070:layer:AWSOpenTelemetryDistroPython:9
Europe (Spain)	arn:aws:lambda:eu-south-2:490004653786:layer:AWSOpenTelemetryDistroPython:6

Region	ARN
Europe (Stockholm)	arn:aws:lambda:eu-north-1:615299751070:layer:AWSOpenTelemetryDistroPython:9
Europe (Zurich)	arn:aws:lambda:eu-central-2:156041407956:layer:AWSOpenTelemetryDistroPython:6
Israel (Tel Aviv)	arn:aws:lambda:il-central-1:746669239226:layer:AWSOpenTelemetryDistroPython:6
Middle East (Bahrain)	arn:aws:lambda:me-south-1:980921751758:layer:AWSOpenTelemetryDistroPython:6
Middle East (UAE)	arn:aws:lambda:me-central-1:739275441131:layer:AWSOpenTelemetryDistroPython:6
South America (São Paulo)	arn:aws:lambda:sa-east-1:615299751070:layer:AWSOpenTelemetryDistroPython:9

Node.js

Region	ARN
US East (N. Virginia)	arn:aws:lambda:us-east-1:615299751070:layer:AWSOpenTelemetryDistroJs:6
US East (Ohio)	arn:aws:lambda:us-east-2:615299751070:layer:AWSOpenTelemetryDistroJs:6
US West (N. California)	arn:aws:lambda:us-west-1:615299751070:layer:AWSOpenTelemetryDistroJs:6
US West (Oregon)	arn:aws:lambda:us-west-2:615299751070:layer:AWSOpenTelemetryDistroJs:6
Africa (Cape Town)	arn:aws:lambda:af-south-1:904233096616:layer:AWSOpenTelemetryDistroJs:6

Region	ARN
Asia Pacific (Hong Kong)	arn:aws:lambda:ap-east-1:888577020596:layer:AWSOpenTelemetryDistroJs:6
Asia Pacific (Hyderabad)	arn:aws:lambda:ap-south-2:796973505492:layer:AWSOpenTelemetryDistroJs:6
Asia Pacific (Jakarta)	arn:aws:lambda:ap-southeast-3:039612877180:layer:AWSOpenTelemetryDistroJs:6
Asia Pacific (Melbourne)	arn:aws:lambda:ap-southeast-4:713881805771:layer:AWSOpenTelemetryDistroJs:6
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:615299751070:layer:AWSOpenTelemetryDistroJs:6
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:615299751070:layer:AWSOpenTelemetryDistroJs:6
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:615299751070:layer:AWSOpenTelemetryDistroJs:6
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:615299751070:layer:AWSOpenTelemetryDistroJs:6
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:615299751070:layer:AWSOpenTelemetryDistroJs:6
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:615299751070:layer:AWSOpenTelemetryDistroJs:6
Canada (Central)	arn:aws:lambda:ca-central-1:615299751070:layer:AWSOpenTelemetryDistroJs:6
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:615299751070:layer:AWSOpenTelemetryDistroJs:6

Region	ARN
Europe (Ireland)	arn:aws:lambda:eu-west-1:615299751070:layer:AWSOpenTelemetryDistroJs:6
Europe (London)	arn:aws:lambda:eu-west-2:615299751070:layer:AWSOpenTelemetryDistroJs:6
Europe (Milan)	arn:aws:lambda:eu-south-1:257394471194:layer:AWSOpenTelemetryDistroJs:6
Europe (Paris)	arn:aws:lambda:eu-west-3:615299751070:layer:AWSOpenTelemetryDistroJs:6
Europe (Spain)	arn:aws:lambda:eu-south-2:490004653786:layer:AWSOpenTelemetryDistroJs:6
Europe (Stockholm)	arn:aws:lambda:eu-north-1:615299751070:layer:AWSOpenTelemetryDistroJs:6
Europe (Zurich)	arn:aws:lambda:eu-central-2:156041407956:layer:AWSOpenTelemetryDistroJs:6
Israel (Tel Aviv)	arn:aws:lambda:il-central-1:746669239226:layer:AWSOpenTelemetryDistroJs:6
Middle East (Bahrain)	arn:aws:lambda:me-south-1:980921751758:layer:AWSOpenTelemetryDistroJs:6
Middle East (UAE)	arn:aws:lambda:me-central-1:739275441131:layer:AWSOpenTelemetryDistroJs:6
South America (São Paulo)	arn:aws:lambda:sa-east-1:615299751070:layer:AWSOpenTelemetryDistroJs:6

.Net

Region	ARN
US East (N. Virginia)	arn:aws:lambda:us-east-1:615299751070:layer:AWSOpenTelemetryDistroDotNet:4
US East (Ohio)	arn:aws:lambda:us-east-2:615299751070:layer:AWSOpenTelemetryDistroDotNet:3
US West (N. California)	arn:aws:lambda:us-west-1:615299751070:layer:AWSOpenTelemetryDistroDotNet:3
US West (Oregon)	arn:aws:lambda:us-west-2:615299751070:layer:AWSOpenTelemetryDistroDotNet:3
Africa (Cape Town)	arn:aws:lambda:af-south-1:904233096616:layer:AWSOpenTelemetryDistroDotNet:3
Asia Pacific (Hong Kong)	arn:aws:lambda:ap-east-1:888577020596:layer:AWSOpenTelemetryDistroDotNet:3
Asia Pacific (Hyderabad)	arn:aws:lambda:ap-south-2:796973505492:layer:AWSOpenTelemetryDistroDotNet:3
Asia Pacific (Jakarta)	arn:aws:lambda:ap-southeast-3:039612877180:layer:AWSOpenTelemetryDistroDotNet:3
Asia Pacific (Melbourne)	arn:aws:lambda:ap-southeast-4:713881805771:layer:AWSOpenTelemetryDistroDotNet:3
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:615299751070:layer:AWSOpenTelemetryDistroDotNet:3
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:615299751070:layer:AWSOpenTelemetryDistroDotNet:3
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:615299751070:layer:AWSOpenTelemetryDistroDotNet:3

Region	ARN
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:615299751070:layer:AWSOpenTelemetryDistroDotNet:3
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:615299751070:layer:AWSOpenTelemetryDistroDotNet:3
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:615299751070:layer:AWSOpenTelemetryDistroDotNet:3
Canada (Central)	arn:aws:lambda:ca-central-1:615299751070:layer:AWSOpenTelemetryDistroDotNet:3
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:615299751070:layer:AWSOpenTelemetryDistroDotNet:3
Europe (Ireland)	arn:aws:lambda:eu-west-1:615299751070:layer:AWSOpenTelemetryDistroDotNet:3
Europe (London)	arn:aws:lambda:eu-west-2:615299751070:layer:AWSOpenTelemetryDistroDotNet:3
Europe (Milan)	arn:aws:lambda:eu-south-1:257394471194:layer:AWSOpenTelemetryDistroDotNet:3
Europe (Paris)	arn:aws:lambda:eu-west-3:615299751070:layer:AWSOpenTelemetryDistroDotNet:3
Europe (Spain)	arn:aws:lambda:eu-south-2:490004653786:layer:AWSOpenTelemetryDistroDotNet:3
Europe (Stockholm)	arn:aws:lambda:eu-north-1:615299751070:layer:AWSOpenTelemetryDistroDotNet:3
Europe (Zurich)	arn:aws:lambda:eu-central-2:156041407956:layer:AWSOpenTelemetryDistroDotNet:3

Region	ARN
Israel (Tel Aviv)	arn:aws:lambda:il-central-1:746669239226:layer:AWSOpenTelemetryDistroDotNet:3
Middle East (Bahrain)	arn:aws:lambda:me-south-1:980921751758:layer:AWSOpenTelemetryDistroDotNet:3
Middle East (UAE)	arn:aws:lambda:me-central-1:739275441131:layer:AWSOpenTelemetryDistroDotNet:3
South America (São Paulo)	arn:aws:lambda:sa-east-1:615299751070:layer:AWSOpenTelemetryDistroDotNet:3

Java

Region	ARN
US East (N. Virginia)	arn:aws:lambda:us-east-1:615299751070:layer:AWSOpenTelemetryDistroJava:3
US East (Ohio)	arn:aws:lambda:us-east-2:615299751070:layer:AWSOpenTelemetryDistroJava:3
US West (N. California)	arn:aws:lambda:us-west-1:615299751070:layer:AWSOpenTelemetryDistroJava:3
US West (Oregon)	arn:aws:lambda:us-west-2:615299751070:layer:AWSOpenTelemetryDistroJava:3
Africa (Cape Town)	arn:aws:lambda:af-south-1:904233096616:layer:AWSOpenTelemetryDistroJava:3
Asia Pacific (Hong Kong)	arn:aws:lambda:ap-east-1:888577020596:layer:AWSOpenTelemetryDistroJava:3
Asia Pacific (Hyderabad)	arn:aws:lambda:ap-south-2:796973505492:layer:AWSOpenTelemetryDistroJava:3

Region	ARN
Asia Pacific (Jakarta)	arn:aws:lambda:ap-southeast-3:039612877180:layer:AWSOpenTelemetryDistroJava:3
Asia Pacific (Melbourne)	arn:aws:lambda:ap-southeast-4:713881805771:layer:AWSOpenTelemetryDistroJava:3
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:615299751070:layer:AWSOpenTelemetryDistroJava:3
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:615299751070:layer:AWSOpenTelemetryDistroJava:3
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:615299751070:layer:AWSOpenTelemetryDistroJava:3
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:615299751070:layer:AWSOpenTelemetryDistroJava:3
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:615299751070:layer:AWSOpenTelemetryDistroJava:3
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:615299751070:layer:AWSOpenTelemetryDistroJava:3
Canada (Central)	arn:aws:lambda:ca-central-1:615299751070:layer:AWSOpenTelemetryDistroJava:3
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:615299751070:layer:AWSOpenTelemetryDistroJava:3
Europe (Ireland)	arn:aws:lambda:eu-west-1:615299751070:layer:AWSOpenTelemetryDistroJava:3
Europe (London)	arn:aws:lambda:eu-west-2:615299751070:layer:AWSOpenTelemetryDistroJava:3

Region	ARN
Europe (Milan)	arn:aws:lambda:eu-south-1:257394471194:layer:AWSOpenTelemetryDistroJava:3
Europe (Paris)	arn:aws:lambda:eu-west-3:615299751070:layer:AWSOpenTelemetryDistroJava:3
Europe (Spain)	arn:aws:lambda:eu-south-2:490004653786:layer:AWSOpenTelemetryDistroJava:3
Europe (Stockholm)	arn:aws:lambda:eu-north-1:615299751070:layer:AWSOpenTelemetryDistroJava:3
Europe (Zurich)	arn:aws:lambda:eu-central-2:156041407956:layer:AWSOpenTelemetryDistroJava:3
Israel (Tel Aviv)	arn:aws:lambda:il-central-1:746669239226:layer:AWSOpenTelemetryDistroJava:3
Middle East (Bahrain)	arn:aws:lambda:me-south-1:980921751758:layer:AWSOpenTelemetryDistroJava:3
Middle East (UAE)	arn:aws:lambda:me-central-1:739275441131:layer:AWSOpenTelemetryDistroJava:3
South America (São Paulo)	arn:aws:lambda:sa-east-1:615299751070:layer:AWSOpenTelemetryDistroJava:3

Deploy Lambda functions using Amazon ECR container

Lambda functions deployed as container images do not support Lambda Layers in the traditional way. When using container images, you cannot attach a layer as you would with other Lambda deployment methods. Instead, you must manually incorporate the layer's contents into your container image during the build process.

Java

You can learn how to integrate the Amazon Lambda Layer for OpenTelemetry into your containerized Java Lambda function, download the `layer.zip` artifact, and integrate it into your Java Lambda function container to enable Application Signals monitoring.

Prerequisites

- Amazon CLI configured with your credentials
- Docker installed
- These instructions assume you are on x86_64 platform

1. **Set Up Project Structure**

   Create a directory for your Lambda function

   ```
   mkdir java-appsignals-container-lambda && \
   cd java-appsignals-container-lambda
   ```

   Create a Maven project structure

   ```
   mkdir -p src/main/java/com/example/java/lambda
   mkdir -p src/main/resources
   ```

2. **Create Dockerfile**

   Download and integrate the OpenTelemetry Layer with Application Signals support directly into your Lambda container image. To do this, the `Dockerfile` file is created.

   ```
   FROM public.ecr.aws/lambda/java:21

   # Install utilities
   RUN dnf install -y unzip wget maven

   # Download the OpenTelemetry Layer with AppSignals Support
   RUN wget https://github.com/aws-observability/aws-otel-java-instrumentation/
   releases/latest/download/layer.zip -O /tmp/layer.zip

   # Extract and include Lambda layer contents
   RUN mkdir -p /opt && \
   ```

```
    unzip /tmp/layer.zip -d /opt/ && \
    chmod -R 755 /opt/ && \
    rm /tmp/layer.zip

# Copy and build function code
COPY pom.xml ${LAMBDA_TASK_ROOT}
COPY src ${LAMBDA_TASK_ROOT}/src
RUN mvn clean package -DskipTests

# Copy the JAR file to the Lambda runtime directory (from inside the container)
RUN mkdir -p ${LAMBDA_TASK_ROOT}/lib/
RUN cp ${LAMBDA_TASK_ROOT}/target/function.jar ${LAMBDA_TASK_ROOT}/lib/

# Set the handler
CMD ["com.example.java.lambda.App::handleRequest"]
```

> **Note**
>
> The `layer.zip` file contains the OpenTelemetry instrumentation necessary for Amazon Application Signals support to monitor your Lambda function.
> The layer extraction steps ensures:
>
> - The layer.zip contents are properly extracted to the `/opt/ directory`
> - The `otel-instrument` script receives proper execution permissions
> - The temporary layer.zip file is removed to keep the image size smaller

3. **Lambda function code** – Create a Java file for your Lambda handler at `src/main/java/com/example/lambda/App.java:`

   Your project should look something like:

```
.
### Dockerfile
### pom.xml
### src
    ### main
        ### java
        #   ### com
        #       ### example
        #           ### java
        #               ### lambda
```

```
        #                    ### App.java
        ### resources
```

4. **Build and deploy the container image**

   Set up environment variables

```
AWS_ACCOUNT_ID=$(aws sts get-caller-identity --query Account --output text)
AWS_REGION=$(aws configure get region)

# For fish shell users:
# set AWS_ACCOUNT_ID (aws sts get-caller-identity --query Account --output
 text)
# set AWS_REGION (aws configure get region)
```

   Authenticate with ECR

   First with public ECR (for base image):

```
aws ecr-public get-login-password --region us-east-1 | docker login --username
 AWS --password-stdin public.ecr.aws
```

   Then with your private ECR:

```
aws ecr get-login-password --region $AWS_REGION | docker login --username AWS --
password-stdin $AWS_ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com
```

   Build, tag and push your image

```
# Build the Docker image
docker build -t lambda-appsignals-demo .

# Tag the image
docker tag lambda-appsignals-demo:latest $AWS_ACCOUNT_ID.dkr.ecr.
$AWS_REGION.amazonaws.com/lambda-appsignals-demo:latest

# Push the image
docker push $AWS_ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com/lambda-appsignals-
demo:latest
```

5. **Create and configure the Lambda function**

Create a new function using the Lambda console.

Select **Container image** as the deployment option.

Choose **Browse images** to select your Amazon ECR image.

6. **Testing and verifications – Test your Lambda with a simple event. If the layer integration is successful, your Lambda appears under the Application Signals service map.**

   You will see traces and metrics for your Lambda function in the CloudWatch console.

Troubleshooting

If Application Signals is not working, check the following:

- Check the function logs for any errors related to the OpenTelemetry instrumentation
- Verify if the environment variable `AWS_LAMBDA_EXEC_WRAPPER` is set correctly
- Make sure the layer extraction in the Docker file completed successfully
- Confirm if the IAM permissions are properly attached
- If needed, increase the *Timeout and Memory* settings in the general configuration of the Lambda function

.Net

You can learn how to integrate the OpenTelemetry Layer with Application Signals support into your containerized .Net Lambda function, download the `layer.zip` artifact, and integrate it into your .Net Lambda function to enable Application Signals monitoring.

Prerequisites

- Amazon CLI configured with your credentials
- Docker installed
- .Net 8 SDK
- These instructions assume you are on x86_64 platform

1. **Set Up Project Structure**

   Create a directory for your Lambda function container image

```
mkdir dotnet-appsignals-container-lambda && \
cd dotnet-appsignals-container-lambda
```

2. **Create Dockerfile**

   Download and integrate the OpenTelemetry Layer with Application Signals support directly into your Lambda container image. To do this, the `Dockerfile` file is created.

```
FROM public.ecr.aws/lambda/dotnet:8

# Install utilities
RUN dnf install -y unzip wget dotnet-sdk-8.0 which

# Add dotnet command to docker container's PATH
ENV PATH="/usr/lib64/dotnet:${PATH}"

# Download the OpenTelemetry Layer with AppSignals Support
RUN wget https://github.com/aws-observability/aws-otel-dotnet-instrumentation/
releases/latest/download/layer.zip -O /tmp/layer.zip

# Extract and include Lambda layer contents
RUN mkdir -p /opt && \
    unzip /tmp/layer.zip -d /opt/ && \
    chmod -R 755 /opt/ && \
    rm /tmp/layer.zip

WORKDIR ${LAMBDA_TASK_ROOT}

# Copy the project files
COPY dotnet-lambda-function/src/dotnet-lambda-function/*.csproj
 ${LAMBDA_TASK_ROOT}/
COPY dotnet-lambda-function/src/dotnet-lambda-function/Function.cs
 ${LAMBDA_TASK_ROOT}/
COPY dotnet-lambda-function/src/dotnet-lambda-function/aws-lambda-tools-
defaults.json ${LAMBDA_TASK_ROOT}/

# Install dependencies and build the application
RUN dotnet restore

# Use specific runtime identifier and disable ReadyToRun optimization
RUN dotnet publish -c Release -o out --self-contained false /
p:PublishReadyToRun=false
```

```
# Copy the published files to the Lambda runtime directory
RUN cp -r out/* ${LAMBDA_TASK_ROOT}/

CMD ["dotnet-lambda-function::dotnet_lambda_function.Function::FunctionHandler"]
```

> **Note**
>
> The `layer.zip` file contains the OpenTelemetry instrumentation necessary for Amazon Application Signals support to monitor your Lambda function.
> The layer extraction steps ensures:
>
> - The layer.zip contents are properly extracted to the `/opt/ directory`
> - The `otel-instrument` script receives proper execution permissions
> - The temporary layer.zip file is removed to keep the image size smaller

3. **Lambda function code** – Initialize your Lambda project using the Amazon Lambda .NET template:

```
# Install the Lambda templates if you haven't already
dotnet new -i Amazon.Lambda.Templates

# Create a new Lambda project
dotnet new lambda.EmptyFunction -n dotnet-lambda-function
```

Your project should look something like:

```
.
### Dockerfile
### dotnet-lambda-function
    ### src
    #   ### dotnet-lambda-function
    #       ### Function.cs
    #       ### Readme.md
    #       ### aws-lambda-tools-defaults.json
    #       ### dotnet-lambda-function.csproj
    ### test
        ### dotnet-lambda-function.Tests
            ### FunctionTest.cs
            ### dotnet-lambda-function.Tests.csproj
```

4. **Build and deploy the container image**

   Set up environment variables

```
AWS_ACCOUNT_ID=$(aws sts get-caller-identity --query Account --output text)
AWS_REGION=$(aws configure get region)

# For fish shell users:
# set AWS_ACCOUNT_ID (aws sts get-caller-identity --query Account --output
 text)
# set AWS_REGION (aws configure get region)
```

   Update the `Function.cs` code to:

   Update the `dotnet-lambda-function.csproj` code to:

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <TargetFramework>net8.0>/TargetFramework>
   <ImplicitUsings>enable</ImplicitUsings>
    <Nullable>enable</Nullable>
    <GenerateRuntimeConfigurationFiles>true</GenerateRuntimeConfigurationFiles>
    <AWSProjectType>Lambda</AWSProjectType>

    <CopyLocalLockFileAssemblies>true</CopyLocalLockFileAssemblies>

    <PublishReadyToRun>true</PublishReadyToRun>
 </PropertyGroup>
  <ItemGroup>
    <PackageReference Include="Amazon.Lambda.Core" Version="2.5.0" />
    <PackageReference Include="Amazon.Lambda.Serialization.SystemTextJson"
 Version="2.4.4" />
    <PackageReference Include="AWSSDK.S3" Version="3.7.305.23" />
  </ItemGroup>
</Project>
```

5. **Build and deploy the container image**

   Set up environment variables

```
AWS_ACCOUNT_ID=$(aws sts get-caller-identity --query Account --output text)
AWS_REGION=$(aws configure get region)
```

```
# For fish shell users:
# set AWS_ACCOUNT_ID (aws sts get-caller-identity --query Account --output
 text)
# set AWS_REGION (aws configure get region)
```

Authenticate with public Amazon ECR

```
aws ecr-public get-login-password --region us-east-1 | docker login --username
 AWS --password-stdin public.ecr.aws
```

Authenticate with private Amazon ECR

```
aws ecr get-login-password --region $AWS_REGION | docker login --username AWS --
password-stdin $AWS_ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com
```

Create Amazon ECR repository (if needed)

```
aws ecr create-repository \
    --repository-name lambda-appsignals-demo \
    --region $AWS_REGION
```

Build, tag, and push your image

```
# Build the Docker image
docker build -t lambda-appsignals-demo .

# Tag the image
docker tag lambda-appsignals-demo:latest $AWS_ACCOUNT_ID.dkr.ecr.
$AWS_REGION.amazonaws.com/lambda-appsignals-demo:latest

# Push the image
docker push $AWS_ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com/lambda-appsignals-
demo:latest

5. Create and Configure the Lambda Function
```

6. **Create and configure the Lambda function**

   Create a new function using the Lambda console.

Select **Container image** as the deployment option.

Choose **Browse images** to select your Amazon ECR image.

7. **Testing and verifications – Test your Lambda with a simple event. If the layer integration is successful, your Lambda appears under the Application Signals service map.**

   You will see traces and metrics for your Lambda function in the CloudWatch console.

Troubleshooting

If Application Signals is not working, check the following:

- Check the function logs for any errors related to the OpenTelemetry instrumentation
- Verify if the environment variable `AWS_LAMBDA_EXEC_WRAPPER` is set correctly
- Make sure the layer extraction in the Docker file completed successfully
- Confirm if the IAM permissions are properly attached
- If needed, increase the *Timeout and Memory* settings in the general configuration of the Lambda function

Node.js

You can learn how to integrate the OpenTelemetry Layer with Application Signals support into your containerized Node.js Lambda function, download the `layer.zip` artifact, and integrate it into your Node.js Lambda function to enable Application Signals monitoring.

Prerequisites

- Amazon CLI configured with your credentials
- Docker installed
- These instructions assume you are on x86_64 platform

1. **Set Up Project Structure**

   Create a directory for your Lambda function container image

```
mkdir nodejs-appsignals-container-lambda &&\
```

```
cd nodejs-appsignals-container-lambda
```

2. **Create Dockerfile**

   Download and integrate the OpenTelemetry Layer with Application Signals support directly into your Lambda container image. To do this, the `Dockerfile` file is created.

```
# Dockerfile
FROM public.ecr.aws/lambda/nodejs:22

# Install utilities
RUN dnf install -y unzip wget

# Download the OpenTelemetry Layer with AppSignals Support
RUN wget https://github.com/aws-observability/aws-otel-js-instrumentation/
releases/latest/download/layer.zip -O /tmp/layer.zip

# Extract and include Lambda layer contents
RUN mkdir -p /opt && \
    unzip /tmp/layer.zip -d /opt/ && \
    chmod -R 755 /opt/ && \
    rm /tmp/layer.zip

# Install npm dependencies
RUN npm init -y
RUN npm install

# Copy function code
COPY *.js ${LAMBDA_TASK_ROOT}/

# Set the CMD to your handler
CMD [ "index.handler" ]
```

> **Note**
>
> The `layer.zip` file contains the OpenTelemetry instrumentation necessary for Amazon Application Signals support to monitor your Lambda function.
> The layer extraction steps ensures:
>
> - The layer.zip contents are properly extracted to the `/opt/ directory`
> - The `otel-instrument` script receives proper execution permissions

- The temporary layer.zip file is removed to keep the image size smaller

3. **Lambda function code**

   Create an `index.js` file with the following content:

```
const { S3Client, ListBucketsCommand } = require('@aws-sdk/client-s3');

// Initialize S3 client
const s3Client = new S3Client({ region: process.env.AWS_REGION });

exports.handler = async function(event, context) {
  console.log('Received event:', JSON.stringify(event, null, 2));
  console.log('Handler initializing:', exports.handler.name);

  const response = {
    statusCode: 200,
    body: {}
  };

  try {
    // List S3 buckets
    const command = new ListBucketsCommand({});
    const data = await s3Client.send(command);

    // Extract bucket names
    const bucketNames = data.Buckets.map(bucket => bucket.Name);

    response.body = {
      message: 'Successfully retrieved buckets',
      buckets: bucketNames
    };

  } catch (error) {
    console.error('Error listing buckets:', error);

    response.statusCode = 500;
    response.body = {
      message: `Error listing buckets: ${error.message}`
    };
  }

  return response;
```

```
};
```

Your project structure should look something like this:

```
.
### Dockerfile
### index.js
```

4. **Build and deploy the container image**

   Set up environment variables

```
AWS_ACCOUNT_ID=$(aws sts get-caller-identity --query Account --output text)
AWS_REGION=$(aws configure get region)

# For fish shell users:
# set AWS_ACCOUNT_ID (aws sts get-caller-identity --query Account --output
 text)
# set AWS_REGION (aws configure get region)
```

   Authenticate with public Amazon ECR

```
aws ecr-public get-login-password --region us-east-1 | docker login --username
 AWS --password-stdin public.ecr.aws
```

   Authenticate with private Amazon ECR

```
aws ecr get-login-password --region $AWS_REGION | docker login --username AWS --
password-stdin $AWS_ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com
```

   Create Amazon ECR repository (if needed)

```
aws ecr create-repository \
    --repository-name lambda-appsignals-demo \
    --region $AWS_REGION
```

   Build, tag, and push your image

```
# Build the Docker image
docker build -t lambda-appsignals-demo .
```

```
# Tag the image
docker tag lambda-appsignals-demo:latest $AWS_ACCOUNT_ID.dkr.ecr.
$AWS_REGION.amazonaws.com/lambda-appsignals-demo:latest

# Push the image
docker push $AWS_ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com/lambda-appsignals-
demo:latest

5. Create and Configure the Lambda Function
```

5. **Create and configure the Lambda function**

   Create a new function using the Lambda console.

   Select **Container image** as the deployment option.

   Choose **Browse images** to select your Amazon ECR image.

6. **Testing and verifications – Test your Lambda with a simple event. If the layer integration is successful, your Lambda appears under the Application Signals service map.**

   You will see traces and metrics for your Lambda function in the CloudWatch console.

Troubleshooting

If Application Signals is not working, check the following:

- Check the function logs for any errors related to the OpenTelemetry instrumentation
- Verify if the environment variable `AWS_LAMBDA_EXEC_WRAPPER` is set correctly
- Make sure the layer extraction in the Docker file completed successfully
- Confirm if the IAM permissions are properly attached
- If needed, increase the *Timeout and Memory* settings in the general configuration of the Lambda function

Python

You can learn how to integrate the OpenTelemetry Layer with Application Signals support into your containerized Python Lambda function, download the `layer.zip` artifact, and integrate it into your Python Lambda function to enable Application Signals monitoring.

Prerequisites

- Amazon CLI configured with your credentials
- Docker installed
- These instructions assume you are on x86_64 platform

1. **Set Up Project Structure**

   Create a directory for your Lambda function container image

   ```
   mkdir python-appsignals-container-lambda &&\
   cd python-appsignals-container-lambda
   ```

2. **Create Dockerfile**

   Download and integrate the OpenTelemetry Layer with Application Signals support directly into your Lambda container image. To do this, the `Dockerfile` file is created.

   > **Note**
   >
   > The `layer.zip` file contains the OpenTelemetry instrumentation necessary for Amazon Application Signals support to monitor your Lambda function.
   > The layer extraction steps ensures:
   >
   > - The layer.zip contents are properly extracted to the `/opt/ directory`
   > - The `otel-instrument` script receives proper execution permissions
   > - The temporary layer.zip file is removed to keep the image size smaller

3. **Lambda function code**

   Create your Lambda function in an `app.py` file:

   ```
   import json
   import boto3

   def lambda_handler(event, context):
       """
       Sample Lambda function that can be used in a container image.
   ```

```
    Parameters:
    -----------
    event: dict
        Input event data
    context: LambdaContext
        Lambda runtime information

    Returns:
    --
    dict
        Response object
    """
    print("Received event:", json.dumps(event, indent=2))

    # Create S3 client
    s3 = boto3.client('s3')

    try:
        # List buckets
        response = s3.list_buckets()

        # Extract bucket names
        buckets = [bucket['Name'] for bucket in response['Buckets']]

        return {
            'statusCode': 200,
            'body': json.dumps({
                'message': 'Successfully retrieved buckets',
                'buckets': buckets
            })
        }
    except Exception as e:
        print(f"Error listing buckets: {str(e)}")
        return {
            'statusCode': 500,
            'body': json.dumps({
                'message': f'Error listing buckets: {str(e)}'
            })
        }
```

Your project structure should look something like this:

```
.
```

```
### Dockerfile
### app.py
### instructions.md
```

4. **Build and deploy the container image**

   Set up environment variables

```
AWS_ACCOUNT_ID=$(aws sts get-caller-identity --query Account --output text)
AWS_REGION=$(aws configure get region)

# For fish shell users:
# set AWS_ACCOUNT_ID (aws sts get-caller-identity --query Account --output
 text)
# set AWS_REGION (aws configure get region)
```

   Authenticate with public Amazon ECR

```
aws ecr-public get-login-password --region us-east-1 | docker login --username
 AWS --password-stdin public.ecr.aws
```

   Authenticate with private Amazon ECR

```
aws ecr get-login-password --region $AWS_REGION | docker login --username AWS --
password-stdin $AWS_ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com
```

   Create Amazon ECR repository (if needed)

```
aws ecr create-repository \
    --repository-name lambda-appsignals-demo \
    --region $AWS_REGION
```

   Build, tag, and push your image

```
# Build the Docker image
docker build -t lambda-appsignals-demo .

# Tag the image
docker tag lambda-appsignals-demo:latest $AWS_ACCOUNT_ID.dkr.ecr.
$AWS_REGION.amazonaws.com/lambda-appsignals-demo:latest
```

```
# Push the image
docker push $AWS_ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com/lambda-appsignals-
demo:latest

5. Create and Configure the Lambda Function
```

5. **Create and configure the Lambda function**

   Create a new function using the Lambda console.

   Select **Container image** as the deployment option.

   Choose **Browse images** to select your Amazon ECR image.

6. **Testing and verifications – Test your Lambda with a simple event. If the layer integration is successful, your Lambda appears under the Application Signals service map.**

   You will see traces and metrics for your Lambda function in the CloudWatch console.

Troubleshooting

If Application Signals is not working, check the following:

- Check the function logs for any errors related to the OpenTelemetry instrumentation
- Verify if the environment variable `AWS_LAMBDA_EXEC_WRAPPER` is set correctly
- Make sure the layer extraction in the Docker file completed successfully
- Confirm if the IAM permissions are properly attached
- If needed, increase the *Timeout and Memory* settings in the general configuration of the Lambda function

Troubleshooting your Application Signals installation

This section contains troubleshooting tips for CloudWatch Application Signals.

Topics

Application Signals Java layer cold start performance

Adding the Application Signals Layer to Java Lambda functions increases the startup latency (cold start time). The following tips can help reduce latency for time-sensitive functions.

Fast startup for Java agent – The Application Signals Java Lambda Layer includes a Fast Startup feature that's turned off by default but can be enabled by setting the OTEL_JAVA_AGENT_FAST_STARTUP_ENABLED variable to true. When enabled, this feature configures the JVM to use tiered compilation level 1 C1 compiler to generate quick optimized native code for faster cold starts. The C1 compiler prioritizes speed at the cost of long-term optimization whereas the C2 compiler provides superior overall performance by profiling data over time.

For more information, see Fast startup for Java agent .

Reduce cold start times with Provisioned Concurrency – Amazon Lambda provisioned concurrency pre-allocates a specified number of function instances, keeping them initialized and ready to handle requests immediately. This reduces cold-start times by eliminating the need to initialize the function environment during execution, ensuring faster and more consistent performance, especially for latency-sensitive workloads. For more information, see Configuring provisioned concurrency for a function .

Optimize startup performance using Lambda SnapStart – Amazon Lambda SnapStart is a feature that optimizes the startup performance of Lambda functions by creating a pre-initialized snapshot of the execution environment after the function's initialization phase. This snapshot is then reused to start new instances, significantly reducing cold-start times by skipping the initialization process during function invocation. For information, see Improving startup performance with Lambda SnapStart

Application doesn't start after Application Signals is enabled

If your application on an Amazon EKS cluster doesn't start after you enable Application Signals on the cluster, check for the following:

- Check if the application has been instrumented by another monitoring solution. Application Signals might not support co-existing with other instrumentation solutions.
- Confirm that your the application meets the compatibility requirements to use Application Signals. For more information, see Supported systems.
- If your application failed to pull the Application Signals artifacts such as the Amazon Distro for OpenTelemetery Java or Python agent and CloudWatch agent images, it could be a network issue.

To mitigate the issue, remove the annotation `instrumentation.opentelemetry.io/inject-java: "true"` or `instrumentation.opentelemetry.io/inject-python: "true"` from your application deployment manifest, and re-deploy your application. Then check if the application is working.

Known issues

The runtime metrics collection in the Java SDK release v1.32.5 is known to not work with applications using JBoss Wildfly. This issue extends to the Amazon CloudWatch Observability EKS add-on, affecting versions `2.3.0-eksbuild.1` through `2.5.0-eksbuild.1`.

If you are impacted, either downgrade the version or disable your runtime metrics collection by adding the environment variable `OTEL_AWS_APPLICATION_SIGNALS_RUNTIME_ENABLED=false` to your application.

Python application doesn't start after Application Signals is enabled

It is a known issue in OpenTelemetry auto-instrumentation that a missing `PYTHONPATH` environment variable can sometimes cause the application to fail to start . To resolve this, ensure that you set the PYTHONPATH environment variable to the location of your application's working directory. For more information about this issue, see [Python autoinstrumentation setting of PYTHONPATH is not compliant with Python's module resolution behavior, breaking Django applications].

For Django applications, there are additional required configurations, which are outlined in the [OpenTelemetry Python documentation].

- Use the `--noreload` flag to prevent automatic reloading.
- Set the `DJANGO_SETTINGS_MODULE` environment variable to the location of your Django application's `settings.py` file. This ensures that OpenTelemetry can correctly access and integrate with your Django settings.

No Application Signals data for Python application that uses a WSGI server

If you are using a WSGI server such as Gunicorn or uWSGI, you must make additional changes to make the ADOT Python auto-instrumentation work.

> **Note**
>
> Be sure that you are using the latest version of ADOT Python and the Amazon CloudWatch Observability EKS add-on before proceeding.

Additional steps to enable Application Signals with a WSGI server

1. Import the auto-instrumentation in the forked worker processes.

   For Gunicorn, use the `post_fork` hook:

```
# gunicorn.conf.py
def post_fork(server, worker):
    from opentelemetry.instrumentation.auto_instrumentation import sitecustomize
```

   For uWSGI, use the `import` directive.

```
#  uwsgi.ini
[uwsgi]
; required for the instrumentation of worker processes
```

```
enable-threads = true
lazy-apps = true
import = opentelemetry.instrumentation.auto_instrumentation.sitecustomize
```

2. Enable the configuration for ADOT Python auto-instrumentation to skip the main process and defer to workers by setting the `OTEL_AWS_PYTHON_DEFER_TO_WORKERS_ENABLED` environment variable to `true`.

My Node.js application is not instrumented or isn't generating Application Signals telemetry

To enable Application Signals for Node.js, you must ensure that your Node.js application uses the CommonJS (CJS) module format. Currently, the Amazon Distro for OpenTelemetry Node.js doesn't support the ESM module format, because OpenTelemetry JavaScript's support of ESM is experimental and is a work in progress.

To determine if your application is using CJS and not ESM, ensure that your application does not fulfill the conditions to enable ESM.

No application data in Application Signals dashboard

If metrics or traces are missing in the Application Signals dashboards, the following might be causes. Investigate these causes only if you have waited 15 minutes for Application Signals to collect and display data since your last update.

- Make sure that your library and framework you are using is supported by the ADOT Java agent. For more information, see Libraries / Frameworks.
- Make sure that the CloudWatch agent is running. First check the status of the CloudWatch agent pods and make sure they are all in `Running` status.

  ```
  kubectl -n amazon-cloudwatch get pods.
  ```

  Add the following to the CloudWatch agent configuration file to enable debugging logs, and then restart the agent.

  ```
  "agent": {

    "region": "${REGION}",
    "debug": true
  ```

```
},
```

Then check for errors in the CloudWatch agent pods.

- Check for configuration issues with the CloudWatch agent. Confirm that the following is still in the CloudWatch agent configuration file and the agent has been restarted since it was added.

  ```
  "agent": {
    "region": "${REGION}",
    "debug": true
  },
  ```

  Then check the OpenTelemetry debugging logs for error messages such as `ERROR io.opentelemetry.exporter.internal.grpc.OkHttpGrpcExporter - Failed to export ...`. These messages might indicate the problem.

  If that doesn't solve the issue, dump and check the environment variables with names that start with `OTEL_` by describing the pod with the `kubectl describe pod` command.

- To enable the OpenTelemetry Python debug logging, set the environment variable `OTEL_PYTHON_LOG_LEVEL` to `debug` and redeploy the application.
- Check for wrong or insufficient permissions for exporting data from the CloudWatch agent. If you see `Access Denied` messages in the CloudWatch agent logs, this might be the issue. It is possible that the permissions applied when you installed the CloudWatch agent were later changed or revoked.
- Check for an Amazon Distro for OpenTelemetry (ADOT) issue when generating telemetry data.

  Make sure that the instrumentation annotations `instrumentation.opentelemetry.io/inject-java` and `sidecar.opentelemetry.io/inject-java` are applied to the application deployment and the value is `true`. Without these, the application pods will not be instrumented even if the ADOT addon is installed correctly.

  Next, check if the `init` container is applied on the application and the `Ready` state is `True`. If the `init` container is not ready, see the status for the reason.

  If the issue persists, enable debug logging on the OpenTelemetry Java SDK by setting the environment variable `OTEL_JAVAAGENT_DEBUG` to true and redeploying the application. Then look for messages that start with `ERROR io.telemetry`.

- The metric/span exporter might be dropping data. To find out, check the application log for messages that include `Failed to export...`
- The CloudWatch agent might be getting throttled when sending metrics or spans to Application Signals. Check for messages indicating throttling in the CloudWatch agent logs.
- Make sure that you've enabled the service discovery setup. You need to do this only once in your Region.

  To confirm this, in the CloudWatch console choose **Application Signals**, **Services**. If Step 1 is not marked **Complete**, choose **Start discovering your services**. Data should start flowing in within five minutes.

Service metrics or dependency metrics have Unknown values

If you see **UnknownService**, **UnknownOperation**, **UnknownRemoteService**, or **UnknownRemoteOperation** for a dependency name or operation in the Application Signals dashboards, check whether the occurrence of data points for the unknown remote service and unknown remote operation are coinciding with their deployments.

- **UnknownService** means that the name of an instrumented application is unknown. If the `OTEL_SERVICE_NAME` environment variable is undefined and `service.name` isn't specified in `OTEL_RESOURCE_ATTRIBUTES`, the service name is set to `UnknownService`. To fix this, specify the service name in `OTEL_SERVICE_NAME` or `OTEL_RESOURCE_ATTRIBUTES`.
- **UnknownOperation** means that the name of an invoked operation is unknown. This occurs when Application Signals is unable to discover an operation name which invokes the remote call, or when the extracted operation name contains high cardinality values.
- **UnknownRemoteService** means that the name of the destination service is unknown. This occurs when the system is unable to extract the destination service name that the remote call accesses.

  One solution is to create a custom span around the function that sends out the request, and add the attribute `aws.remote.service` with the designated value. Another option is to configure the CloudWatch agent to customize the metric value of `RemoteService`. For more information about customizations in the CloudWatch agent, see [Enable CloudWatch Application Signals](#).
- **UnknownRemoteOperation** means that the name of the destination operation is unknown. This occurs when the system is unable to extract the destination operation name that the remote call accesses.

One solution is to create a custom span around the function that sends out the request, and add the attribute `aws.remote.operation` with the designated value. Another option is to configure the CloudWatch agent to customize the metric value of `RemoteOperation`. For more information about customizations in the CloudWatch agent, see Enable CloudWatch Application Signals.

Handling a ConfigurationConflict when managing the Amazon CloudWatch Observability EKS add-on

When you install or update the Amazon CloudWatch Observability EKS add-on, if you notice a failure caused by a `Health Issue` of type `ConfigurationConflict` with a description that starts with `Conflicts found when trying to apply. Will not continue due to resolve conflicts mode`, it is likely because you already have the CloudWatch agent and its associated components such as the ServiceAccount, the ClusterRole and the ClusterRoleBinding installed on the cluster. When the add-on tries to install the CloudWatch agent and its associated components, if it detects any change in the contents, it by default fails the installation or update to avoid overwriting the state of the resources on the cluster.

If you are trying to onboard to the Amazon CloudWatch Observability EKS add-on and you see this failure, we recommend deleting an existing CloudWatch agent setup that you had previously installed on the cluster and then installing the EKS add-on. Be sure to back up any customizations you might have made to the original CloudWatch agent setup such as a custom agent configuration, and provide these to the Amazon CloudWatch Observability EKS add-on when you next install or update it. If you had previously installed the CloudWatch agent for onboarding to Container Insights, see Deleting the CloudWatch agent and Fluent Bit for Container Insights for more information.

Alternatively, the add-on supports a conflict resolution configuration option that has the capability to specify `OVERWRITE`. You can use this option to proceed with installing or updating the add-on by overwriting the conflicts on the cluster. If you are using the Amazon EKS console, you'll find the **Conflict resolution method** when you choose the **Optional configuration settings** when you create or update the add-on. If you are using the Amazon CLI, you can supply the `--resolve-conflicts OVERWRITE` to your command to create or update the add-on.

I want to filter out unnecessary metrics and traces

If Application Signals is collecting traces and metrics that you don't want, see Manage high-cardinality operations for information about configuring the CloudWatch agent with custom rules to reduce cardinality.

For information about customizing trace sampling rules, see Configure sampling rules in the X-Ray documentation.

What does `InternalOperation` mean?

An `InternalOperation` is an operation that is triggered by the application internally rather than by an external invocation. Seeing `InternalOperation` is expected, healthy behavior.

Some typical examples where you would see `InternalOperation` include the following:

- **Preloading on start**– Your application performs an operation named `loadDatafromDB` which reads metadata from a database during the warm up phase. Instead of observing `loadDatafromDB` as a service operation, you'll see it categorized as an `InternalOperation`.
- **Async execution in the background**– Your application subscribes to an event queue, and processes streaming data accordingly whenever there's an update. Each triggered operation will be under `InternalOperation` as a service operation.
- **Retrieving host information from a service registry**– Your application talks to a service registry for service discovery. All interactions with the discovery system are classified as an `InternalOperation`.

How do I enable logging for .NET applications?

To enable logging for .NET applications, configure the following environment variables. For more information about how to configure these environment variables, see Troubleshooting .NET automatic instrumentation issues in the OpenTelemetry documentation.

- `OTEL_LOG_LEVEL`
- `OTEL_DOTNET_AUTO_LOG_DIRECTORY`
- `COREHOST_TRACE`
- `COREHOST_TRACEFILE`

How can I resolve assembly version conflicts in .NET applications?

If you get the following error, see Assembly version conflicts in the OpenTelemetry documentation for resolution steps.

```
Unhandled exception. System.IO.FileNotFoundException: Could not load file or
 assembly 'Microsoft.Extensions.DependencyInjection.Abstractions, Version=7.0.0.0,
 Culture=neutral, PublicKeyToken=adb9793829ddae60'. The system cannot find the file
 specified.

File name: 'Microsoft.Extensions.DependencyInjection.Abstractions, Version=7.0.0.0,
 Culture=neutral, PublicKeyToken=adb9793829ddae60'
   at Microsoft.AspNetCore.Builder.WebApplicationBuilder..ctor(WebApplicationOptions
 options, Action`1 configureDefaults)
   at Microsoft.AspNetCore.Builder.WebApplication.CreateBuilder(String[] args)
   at Program.<Main>$(String[] args) in /Blog.Core/Blog.Core.Api/Program.cs:line 26
```

Can I disable FluentBit?

You can disable FluentBit by configuring the Amazon CloudWatch Observability EKS add-on. For more information, see (Optional) Additional configuration.

Can I filter container logs before exporting to CloudWatch Logs?

No, filtering container logs is not yet supported.

Update to required versions of agents or Amazon EKS add-on

After August 9, 2024, CloudWatch Application Signals will no longer support older versions of the Amazon CloudWatch Observability EKS add-on, the CloudWatch agent, and the Amazon Distro for OpenTelemetry auto-instrumentation agent.

- For the Amazon CloudWatch Observability EKS add-on, versions older than `v1.7.0-eksbuild.1` won't be supported.
- For the CloudWatch agent, versions older than `1.300040.0` won't be supported.
- For the Amazon Distro for OpenTelemetry auto-instrumentation agent:
  - For Java, versions older than `1.32.2` aren't supported.
  - For Python, versions older than `0.2.0` aren't supported.
  - For .NET, versions older than `1.3.2` aren't supported.
  - For Node.js, versions older than `0.3.0` aren't supported.

> **Important**
>
> The latest versions of the agents include updates to the Application Signals metric schema. These updates are not backward compatible, and this can result in data issues if incompatible versions are used. To help ensure a seamless transition to the new functionality, do the following:
>
> - If your application is running on Amazon EKS, be sure to restart all instrumented applications after you update the Amazon CloudWatch Observability add-on.
> - For applications running on other platforms, be sure to upgrade **both** the CloudWatch agent and the Amazon OpenTelemetry auto-instrumentation agent to the latest versions.

The instructions in the following sections can help you update to a supported version.

Contents

Update the Amazon CloudWatch Observability EKS add-on

To the Amazon CloudWatch Observability EKS add-on, you can use the Amazon Web Services Management Console or the Amazon CLI.

Use the console

To upgrade the add-on using the console

1. Open the Amazon EKS console at https://console.amazonaws.cn/eks/home#/clusters.
2. Choose the name of the Amazon EKS cluster to update.
3. Choose the **Add-ons** tab, then choose **Amazon CloudWatch Observability**.

4. Choose **Edit**, select the version you want to update to, and then choose **Save changes**.

   Be sure to choose `v1.7.0-eksbuild.1` or later.

5. Enter one of the following Amazon CLI commands to restart your services.

```
# Restart a deployment
kubectl rollout restart deployment/name
# Restart a daemonset
kubectl rollout restart daemonset/name
# Restart a statefulset
kubectl rollout restart statefulset/name
```

Use the Amazon CLI

To upgrade the add-on using the Amazon CLI

1. Enter the following command to find the latest version.

```
aws eks describe-addon-versions \
--addon-name amazon-cloudwatch-observability
```

2. Enter the following command to update the add-on. Replace *$VERSION* with a version that is `v1.7.0-eksbuild.1` or later. Replace *$AWS_REGION* and *$CLUSTER* with your Region and cluster name.

```
aws eks update-addon \
--region $AWS_REGION \
--cluster-name $CLUSTER \
--addon-name amazon-cloudwatch-observability \
--addon-version $VERSION \
# required only if the advanced configuration is used.
--configuration-values $JSON_CONFIG
```

> **Note**
>
> If you're using an custom configuration for the add-on, you can find an example of the configuration to use for *$JSON_CONFIG* in Enable CloudWatch Application Signals.

3. Enter one of the following Amazon CLI commands to restart your services.

```
# Restart a deployment
kubectl rollout restart deployment/name
# Restart a daemonset
kubectl rollout restart daemonset/name
# Restart a statefulset
kubectl rollout restart statefulset/name
```

Update the CloudWatch agent and ADOT agent

If your services are running on architectures other than Amazon EKS, you will need to upgrade both the CloudWatch agent and the ADOT auto-instrumentation agent to use the latest Application Signals features.

Update on Amazon ECS

To upgrade your agents for services running on Amazon ECS

1. Create a new task definition revision. For more information, see Updating a task definition using the console.
2. Replace the `$IMAGE` of the `ecs-cwagent` container with the latest image tag from cloudwatch-agent on Amazon ECR.

   If you upgrade to a fixed version, be sure to use a version equal to or later than `1.300040.0`.
3. Replace the `$IMAGE` of the `init` container with the latest image tag from the following locations:

   - For Java, use aws-observability/adot-autoinstrumentation-java.

     If you upgrade to a fixed version, be sure to use a version equal to or later than `1.32.2`.
   - For Python, use aws-observability/adot-autoinstrumentation-python.

     If you upgrade to a fixed version, be sure to use a version equal to or later than `0.2.0`.
   - For .NET, use aws-observability/adot-autoinstrumentation-dotnet.

     If you upgrade to a fixed version, be sure to use a version equal to or later than `1.3.2`.
   - For Node.js, use aws-observability/adot-autoinstrumentation-node.

     If you upgrade to a fixed version, be sure to use a version equal to or later than `0.3.0`.

4. Update the Application Signals environment variables in your app container by following the instructions at Step 4: Instrument your application with the CloudWatch agent.
5. Deploy your service with the new task definition.

Update on Amazon EC2 or other architectures

To upgrade your agents for services running on Amazon EC2 or other architectures

1. Follow the instructions at Download the CloudWatch agent package and upgrade the CloudWatch agent to the latest version. Be sure to select version `1.300040.0` or later.
2. Download the latest version of the Amazon Distro for OpenTelemetry auto-instrumentation agent from one of the following locations:

   - For Java, use aws-otel-java-instrumentation .

     If you upgrade to a fixed version, be sure to choose `1.32.2` or later.
   - For Python, use aws-otel-python-instrumentation .

     If you upgrade to a fixed version, be sure to choose `0.2.0` or later.
   - For .NET, use aws-otel-dotnet-instrumentation.

     If you upgrade to a fixed version, be sure to choose `1.3.2` or later.
   - For Node.js, use aws-otel-js-instrumentation .

     If you upgrade to a fixed version, be sure to choose `0.3.0` or later.
3. Apply the updated Application Signals environment variables to your application, then start your application. For more information, see Step 3: Instrument your application and start it.

(Optional) Configuring Application Signals

This section contains information about configuring CloudWatch Application Signals.

Topics

Trace sampling rate

By default, when you enable Application Signals X-Ray centralized sampling is enabled using the default sampling rate settings of `reservoir=1/s` and `fixed_rate=5%`. The environment variables for the Amazon Distro for OpenTelemetry (ADOT) SDK agent as set as follows.

Environment variable	Value	Note
`OTEL_TRACES_SAMPLER`	`xray`	
`OTEL_TRACES_SAMPLER_ARG`	`endpoint=http://cloudwatch-agent.amazon-cloudwatch:2000`	Endpoint of the CloudWatch agent

For information about changing the sampling configuration, see the following:

- To change X-Ray sampling, see [Configure sampling rules](#)
- To change ADOT sampling, see [Configuring the OpenTelemetry Collector for X-Ray remote sampling](#)

If you want to disable X-Ray centralized sampling and use local sampling instead, set the following values for the ADOT SDK Java agent as below. The following example sets the sampling rate at 5%.

Environment variable	Value
`OTEL_TRACES_SAMPLER`	`parentbased_traceidratio`
`OTEL_TRACES_SAMPLER_ARG`	`0.05`

For information about more advanced sampling settings, see [OTEL_TRACES_SAMPLER](#).

Enable trace to log correlation

You can enable *trace to log correlation* in Application Signals. This automatically injects trace IDs and span IDs into the relevant application logs. Then, when you open a trace detail page in the Application Signals console, the relevant log entries (if any) that correlate with the current trace automatically appear at the bottom of the page.

For example, suppose you notice a spike in a latency graph. You can choose the point on the graph to load the diagnostics information for that point in time. You then choose the relevant trace to get more information. When you view the trace information, you can scroll down to see the logs associated with the trace. These logs might reveal patterns or error codes associated with the issues causing the latency spike.

To achieve trace log correlation, Application Signals relies on the following:

- Logger MDC auto-instrumentation for Java.
- OpenTelemetry Logging Instrumentation for Python.
- The Pino, Winston, or Bunyan auto-instrumentations for Node.js.

All of these isntrumentations are provided by OpenTelemetry community. Application Signals uses them to inject trace contexts such as trace ID and span ID into application logs. To enable this, you must manually change your logging configuration to enable the auto-instrumentation.

Depending on the architecture that your application runs on, you might have to also set an environment variable to enable trace log correlation, in addition to following the steps in this section.

- On Amazon EKS, no further steps are needed.
- On Amazon ECS, no further steps are needed.
- On Amazon EC2, see the step 4 in the procedure in Step 3: Instrument your application and start it.

After you enable trace log correlation,

Trace log correlation setup examples

This section contains examples of setting up trace log correlation in several environments.

Spring Boot for Java

Suppose you have a Spring Boot application in a folder called `custom-app`. The application configuration is usually a YAML file named `custom-app/src/main/resources/application.yml` that might look like this:

```
spring:
```

```
  application:
    name: custom-app
  config:
    import: optional:configserver:${CONFIG_SERVER_URL:http://localhost:8888/}

...
```

To enable trace log correlation, add the following logging configuration.

```
spring:
  application:
    name: custom-app
  config:
    import: optional:configserver:${CONFIG_SERVER_URL:http://localhost:8888/}

...

logging:
  pattern:
    level: trace_id=%mdc{trace_id} span_id=%mdc{span_id} trace_flags=%mdc{trace_flags}
 %5p
```

Logback for Java

In the logging configuration (such as logback.xml), insert the trace context `trace_id=%mdc{trace_id} span_id=%mdc{span_id} trace_flags=%mdc{trace_flags} %5p` into `pattern` of Encoder. For example, the following configuration prepends the trace context before the log message.

```
<appender name="FILE" class="ch.qos.logback.core.FileAppender">
  <file>app.log</file>
  <append>true</append>
  <encoder>
    <pattern>trace_id=%mdc{trace_id} span_id=%mdc{span_id} trace_flags=
%mdc{trace_flags} %5p - %m%n</pattern>
  </encoder>
</appender>
```

For more information about encoders in Logback, see [Encoders](#) in the Logback documentation.

Log4j2 for Java

In the logging configuration (such as log4j2.xml), insert the trace context `trace_id=%mdc{trace_id} span_id=%mdc{span_id} trace_flags=%mdc{trace_flags} %5p` into `PatternLayout`. For example, the following configuration prepends the trace context before the log message.

```
<Appenders>
  <File name="FILE" fileName="app.log">
    <PatternLayout pattern="trace_id=%mdc{trace_id} span_id=%mdc{span_id} trace_flags=
%mdc{trace_flags} %5p - %m%n"/>
  </File>
</Appenders>
```

For more information about pattern layouts in Log4j2, see Pattern Layout in the Log4j2 documentation.

Log4j for Java

In the logging configuration (such as log4j.xml), insert the trace context `trace_id=%mdc{trace_id} span_id=%mdc{span_id} trace_flags=%mdc{trace_flags} %5p` into `PatternLayout`. For example, the following configuration prepends the trace context before the log message.

```
<appender name="FILE" class="org.apache.log4j.FileAppender">;
  <param name="File" value="app.log"/>;
  <param name="Append" value="true"/>;
  <layout class="org.apache.log4j.PatternLayout">;
    <param name="ConversionPattern" value="trace_id=%mdc{trace_id} span_id=
%mdc{span_id} trace_flags=%mdc{trace_flags} %5p - %m%n"/>;
  </layout>;
</appender>;
```

For more information about pattern layouts in Log4j, see Class Pattern Layout in the Log4j documentation.

Python

Set the environment variable `OTEL_PYTHON_LOG_CORRELATION` to `true` while running your application. For more information, see Enable trace context injectionin the Python OpenTelemetry documentation.

Node.js

For more information about enabling trace context injection in Node.js for the logging libraries that support it, see the NPM usage documentations of the Pino, Winston, or Bunyan auto-instrumentations for Node.js.

Enable metric to log correlation

If you publish application logs to log groups in CloudWatch Logs, you can enable *metric to application log correlation* in Application Signals. With metric log correlation, the Application Signals console automatically displays the relevant log groups associated with a metric.

For example, suppose you notice a spike in a latency graph. You can choose a point on the graph to load the diagnostics information for that point in time. The diagnostics information will show the relevant application log groups that are associated with the current service and metric. Then you can choose a button to run a CloudWatch Logs Insights query on those log groups. Depending on the information contained in the application logs, this might help you to investigate the cause of the latency spike.

Depending on the architecture that your application runs on, you might have to also set an environment variable to enable metric to application log correlation.

- On Amazon EKS, no further steps are needed.
- On Amazon ECS, no further steps are needed.
- On Amazon EC2, see step 4 in the procedure in Step 3: Instrument your application and start it.

Manage high-cardinality operations

Application Signals includes settings in the CloudWatch agent that you can use to manage the cardinality of your operations and manage the metric exportation to optimize costs. By default, the metric limiting function becomes active when the number of distinct operations for a service over time exceeds the default threshold of 500. You can tune the behavior by adjusting the configuration settings.

Determine if metric limiting is activated

You can use the following methods to find if the default metric limiting is happening. If it is, you should consider optimizing the cardinality control by following the steps in the next section.

- In the CloudWatch console, choose **Application Signals**, **Services**. If you see an **Operation** named **AllOtherOperations** or a **RemoteOperation** named **AllOtherRemoteOperations**, then metric limiting is happening.
- If any metrics collected by Application Signals have the value `AllOtherOperations` for their `Operation` dimension, then metric limiting is happening.
- If any metrics collected by Application Signals have the value `AllOtherRemoteOperations` for their `RemoteOperation` dimension, then metric limiting is happening.

Optimize cardinality control

To optimize your cardinality control, you can do the following:

- Create custom rules to aggregate operations.
- Configure your metric limiting policy.

Create custom rules to aggregate operations

High-cardinality operations can sometimes be caused by inappropriate unique values extracted from the context. For example, sending out HTTP/S requests that include user IDs or session IDs in the path can lead to hundreds of disparate operations. To resolve such issues, we recommend that you configure the CloudWatch agent with customization rules to rewrite these operations.

In cases where there is a surge in generating numerous different metrics through individual `RemoteOperation` calls, such as `PUT /api/customer/owners/123`, `PUT /api/customer/owners/456`, and similar requests, we recommend that you consolidate these operations into a single `RemoteOperation`. One approach is to standardize all `RemoteOperation` calls that start with `PUT /api/customer/owners/` to a uniform format, specifically `PUT /api/customer/owners/{ownerId}`. The following example illustrates this. For information about other customization rules, see Enable CloudWatch Application Signals.

```
{
   "logs":{
      "metrics_collected":{
         "application_signals":{
            "rules":[
               {
                  "selectors":[
                     {
```

```
                        "dimension":"RemoteOperation",
                        "match":"PUT /api/customer/owners/*"
                     }
                  ],
                  "replacements":[
                     {
                        "target_dimension":"RemoteOperation",
                        "value":"PUT /api/customer/owners/{ownerId}"
                     }
                  ],
                  "action":"replace"
               }
            ]
         }
      }
   }
}
```

In other cases, high-cardinality metrics might have been aggregated to `AllOtherRemoteOperations`, and it might be unclear what specific metrics are included. The CloudWatch agent is able to log the dropped operations. To identify dropped operations, use the configuration in the following example to activate logging until the problem resurfaces. Then inspect the CloudWatch agent logs (accessible by container `stdout` or EC2 log files) and search for the keyword `drop metric data`.

```
{
  "agent": {
    "config": {
      "agent": {
        "debug": true
      },
      "traces": {
        "traces_collected": {
          "application_signals": {
          }
        }
      },
      "logs": {
        "metrics_collected": {
          "application_signals": {
            "limiter": {
              "log_dropped_metrics": true
```

```
                }
              }
            }
          }
        }
      }
    }
```

Create your metric limiting policy

If the default metric limiting configuration doesn't address the cardinality for your service, you can customize the metric limiter configuration. To do this, add a `limiter` section under the `logs/metrics_collected/application_signals` section in the CloudWatch Agent configuration file.

The following example lowers the threshold of metric limiting from 500 distinct metrics to 100.

```
{
  "logs": {
    "metrics_collected": {
      "application_signals": {
        "limiter": {
          "drop_threshold": 100
        }
      }
    }
  }
}
```

Monitor the operational health of your applications with Application Signals

Use Application Signals within the CloudWatch console to monitor and troubleshoot the operational health of your applications:

- **Monitor your application services** — As part of daily operational monitoring, use the Services page to see a summary of all your services. See services with the highest fault rate or latency, and see which services have unhealthy service level indicators (SLIs). Select a service to open the Service detail page and see detailed metrics, service operations, Synthetics canaries, and client requests. This can help you troubleshoot and identify the root cause of operational issues.

- **Inspect your application topology** — Use the Service Map to understand and monitor your application topology over time, including the relationships between clients, Synthetics canaries, services, and dependencies. Instantly see service level indicator (SLI) health and view key metrics such as call volume, fault rate, and latency. Drill down to see more detailed information in the Service detail page.

Explore an example scenario that demonstrates how these pages can be used to quickly troubleshoot an operational service health issue, from initial detection to identifying root cause.

How Application Signals enables operational health monitoring

After you enable your application for Application Signals, your application services, APIs, and their dependencies are automatically discovered and displayed in the **Services**, **Service detail**, and **Service Map** pages. Application Signals collects information from multiple sources to enable service discovery and operational health monitoring:

- Amazon Distro for OpenTelemetry (ADOT) — As part of enabling Application Signals, OpenTelemetry Java and Python auto-instrumentation libraries are configured to emit metrics and traces that are collected by the CloudWatch agent. The metrics and traces are used to enable discovery of services, operations, dependencies, and other service information.
- Service-level objectives (SLOs) — After you create service level objectives for your services, the Services, Service detail, and Service Map pages display service level indicator (SLI) health. SLIs can monitor latency, availability, and other operational metrics.
- CloudWatch Synthetics canaries — When you configure X-Ray tracing on your canaries, calls to your services from your canary scripts are associated with your service and displayed within the Service detail page.
- CloudWatch Real user monitoring (RUM) — When X-Ray tracing is enabled on your CloudWatch RUM web client, requests to your services are automatically associated and displayed within the service detail page.
- Amazon Service Catalog AppRegistry — Application Signals auto-discovers Amazon resources within your account and allows you to group them into logical applications created in AppRegistry. The application name displayed in the Services page is based on the underlying compute resource that your services are running on.

> **Note**
>
> Application Signals displays your services and operations based on metrics and traces emitted within the current time filter that you chose. (By default, this is the past three hours.) If there is no activity within the current time filter for a service, operation, dependency, Synthetics canary, or client page, it won't be displayed.
> Currently, up to 1,000 services can be displayed. Discovery of your services and service topology might be delayed up to 10 minutes. Evaluation of your service level indicator (SLI) health might be delayed up to 15 minutes.

> **Note**
>
> Application Signals console currently only supports choosing a maximum of one day within the 30 days time range.

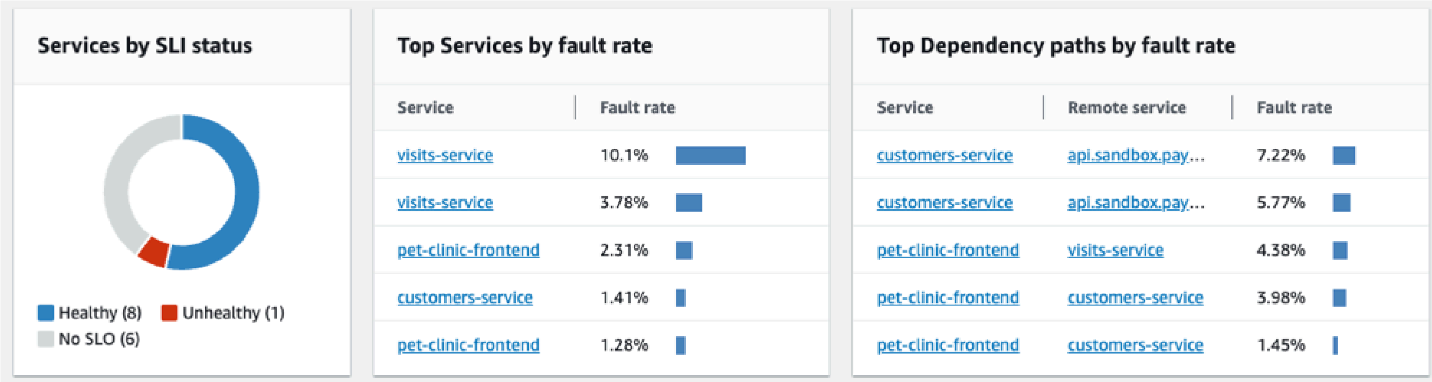
View overall service activity and operational health with the Services page

Use the Services page to see a list of your services that are enabled for Application Signals. You can also view operational metrics and quickly see which services have unhealthy service level indicators (SLIs). Drill down to look for performance anomalies as you identify the root cause of operational issues. To view this page, open the CloudWatch console and choose **Services** under the **Application Signals** section in the left navigation pane.

Explore operational health metrics for your services

The top of the Services page includes an overall service operational health graph and several tables displaying top services and service dependencies by fault rate. The Services graph on the left displays a breakdown of the number of services that have healthy or unhealthy service level indicators (SLIs) during the current page-level time filter. SLIs can monitor latency, availability, and other operational metrics.

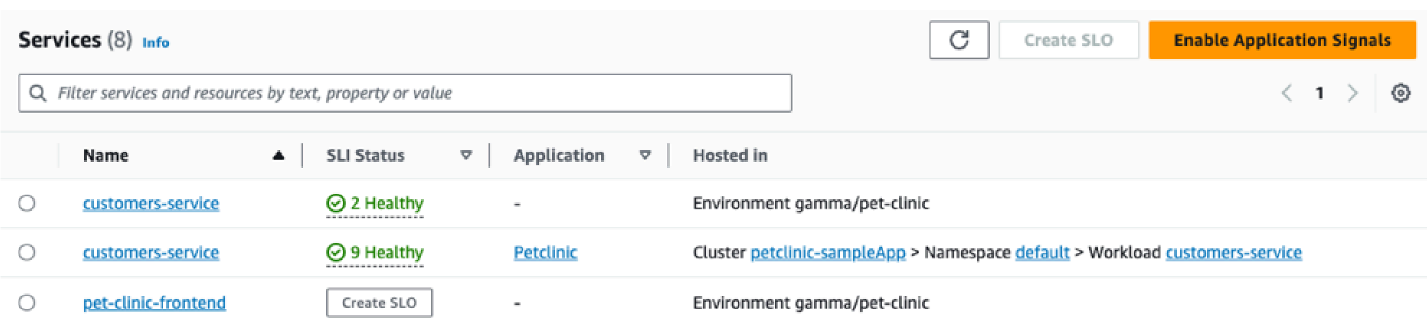
The two tables next to the graph display a list of top services by fault rate. Choose any service name in either table to open a service detail page and see detailed service operation details. Choose a dependency path to open the detail page and see service dependency details. Both tables display information for up to the past three hours, even if a longer time period filter is chosen at the top right of the page.



Monitor operational health with the Services table

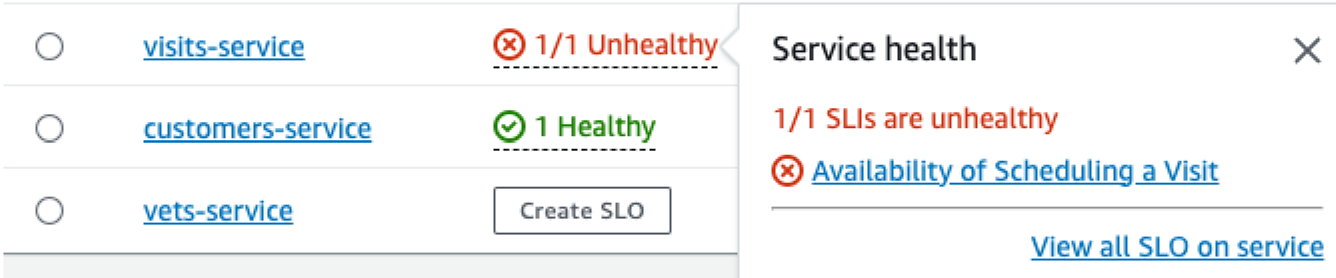
The Services table displays a list of your services that have been enabled for Application Signals. Choose **Enable Application Signals** to open a setup page and start configuring your services. For more information, see Enable Application Signals.

Filter the Services table to make it easier to find what you're looking for, by choosing one or more properties from the filter text box. As you choose each property, you are guided through filter criteria. You will see the complete filter below the filter text box. Choose **Clear filters** at any time to remove the table filter.



Choose the name of any service in the table to view a service detail page containing service-level metrics, operations, and additional details. If you have associated the service's underlying compute resource with an application in AppRegistry or the Applications card on the Amazon Web Services Management Console home page, choose the application name to display the application details in the myApplications console page. For services hosted in Amazon EKS, choose any link within the **Hosted in** column to view Cluster, Namespace, or Workload within CloudWatch Container Insights. For services running on Amazon ECS or Amazon EC2, the Environment value is shown.

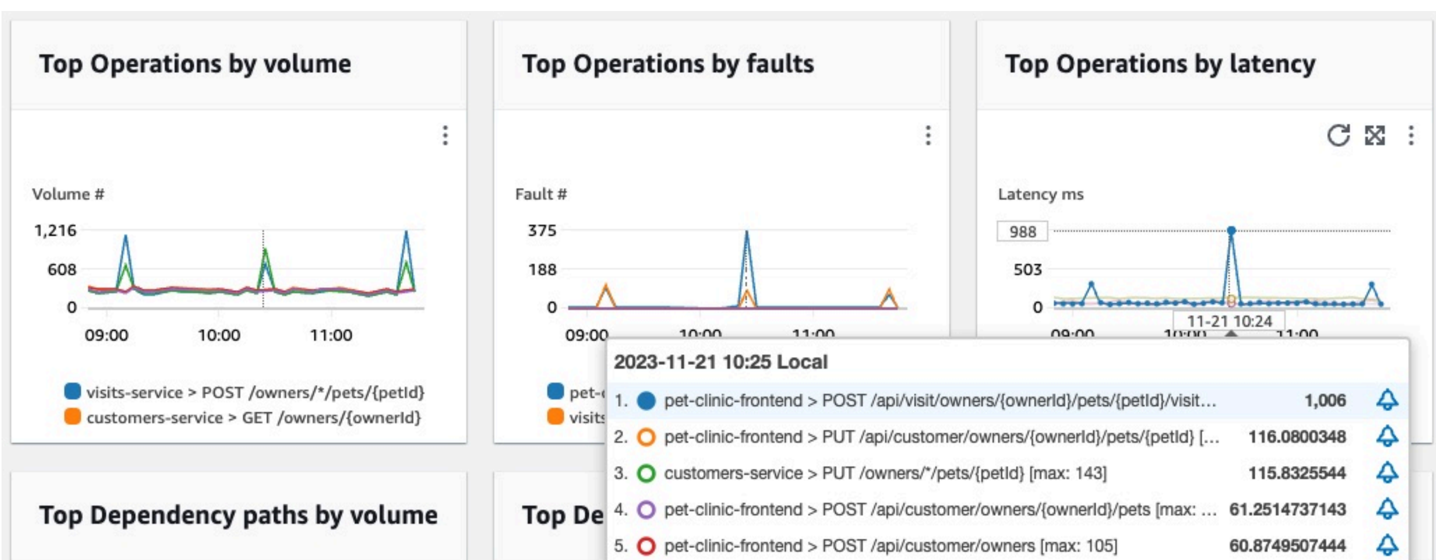
Service level indicator (SLI) status is displayed for each service in the table. Choose the SLI status for a service to display a pop-up containing a link to any unhealthy SLIs, and a link to see all SLOs for the service.



If no SLOs have been created for a service, choose the **Create SLO** button within the **SLI Status** column. To create additional SLOs for any service, select the option button next to the service name, and then choose **Create SLO** at the top-right of the table. When you create SLOs, you can see at a glance which of your services and operations are performing well and which are unhealthy. See service level objectives (SLOs) for more information.

View top operation and dependency metrics

Below the Services table, you can view top operations and dependencies across all services by call volume, faults, and latency. This set of graphs gives you critical information about which operations or dependencies may be unhealthy across all services. Choose any point in a graph to see a pop-up containing more detailed series information. Hover over the series descriptions at the bottom of a graph to see a pop-up containing detailed metrics for a specific operation or dependency path. Select the context menu button at the top-right corner of a graph to see additional options, including viewing CloudWatch metrics or logs pages.



View detailed service activity and operational health with the service detail page

When you instrument your application, Amazon CloudWatch Application Signals maps all of the services that your application discovers. Use the service detail page to see an overview of your services, operations, dependencies, canaries, and client requests for a single service. To view the service detail page, do the following:

- Open the CloudWatch console.
- Choose **Services** under the **Application Signals** section in the left navigation pane.
- Choose the name of any service from the **Services**, **Top services**, or dependency tables.

Under **schedule-visits**, you will see the account label and ID under the service name.

The service detail page is organized into the following tabs:

- Overview — Use this tab to see an overview of a single service, including the number of operations, dependencies, synthetics, and client pages. The tab shows key metrics for your entire service, top operations and dependencies. These metrics include time series data on latency, faults, and errors across all service operations for that service.
- Service operations — Use this tab to see a list of the operations that your service calls and interactive graphs with key metrics that measure the health of each operation. You can select a data point in a graph to obtain information about traces, logs, or metrics associated with that data point.
- Dependencies — Use this tab to see a list of dependencies that your service calls, and a list of metrics for those dependencies.
- Synthetics canaries — Use this tab to see a list of synthetics canaries that simulate user calls to your service, and key performance metrics for how those canaries.
- Client pages — Use this tab to see a list of client pages that call your service, and metrics that measure the quality of client interactions with your application.

View your service overview

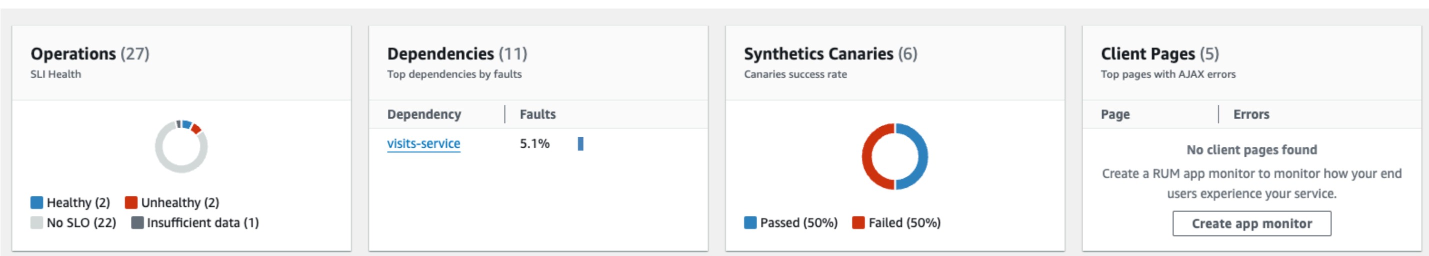
Use the service overview page to view a high-level summary of metrics for all service operations in a single location. Check the performance of all the operations, dependencies, client pages and synthetics canaries that interact with your application. Use this information to help you determine where to focus efforts to identify issues, troubleshoot errors, and find opportunities for optimization.

Choose any link in **Service Details** to view information that is related to a specific service. For example, for services hosted in Amazon EKS, the service details page shows **Cluster**, **Namespace**, and **Workload** information. For services hosted in Amazon ECS or Amazon EC2, the service details page shows the **Environment** value.

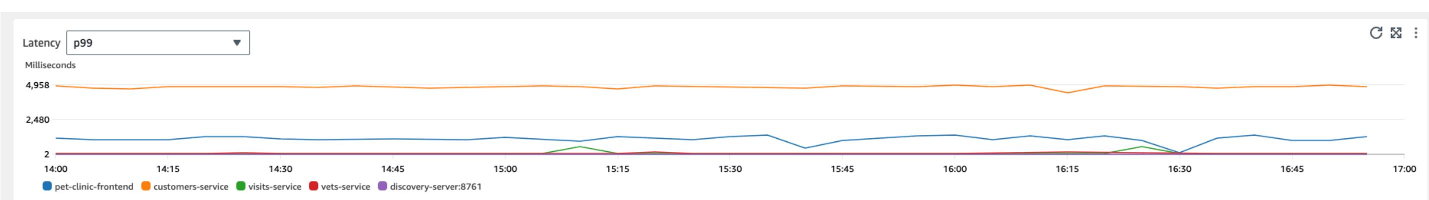
Under **Services**, the **Overview** tab displays a summary of the following:

- Operations – Use this tab to see the health of your service operations. The health status is determined by service level indicators (SLI) that are defined as a part of a service level objective (SLO).
- Dependencies – Use this tab to see the top dependencies of the services called by your application, listed by fault rate and to see the health of your service dependencies. The health status is determined by service level indicators (SLI) that are defined as a part of a service level objective (SLO).
- Synthetics canaries – Use this tab to see the result of simulated calls to endpoints or APIs associated with your service, and the number of failed canaries.
- Client pages – Use this tab to see top pages called by clients that have asynchronous JavaScript and XML (AJAX) errors.

The following illustration shows an overview of your services:

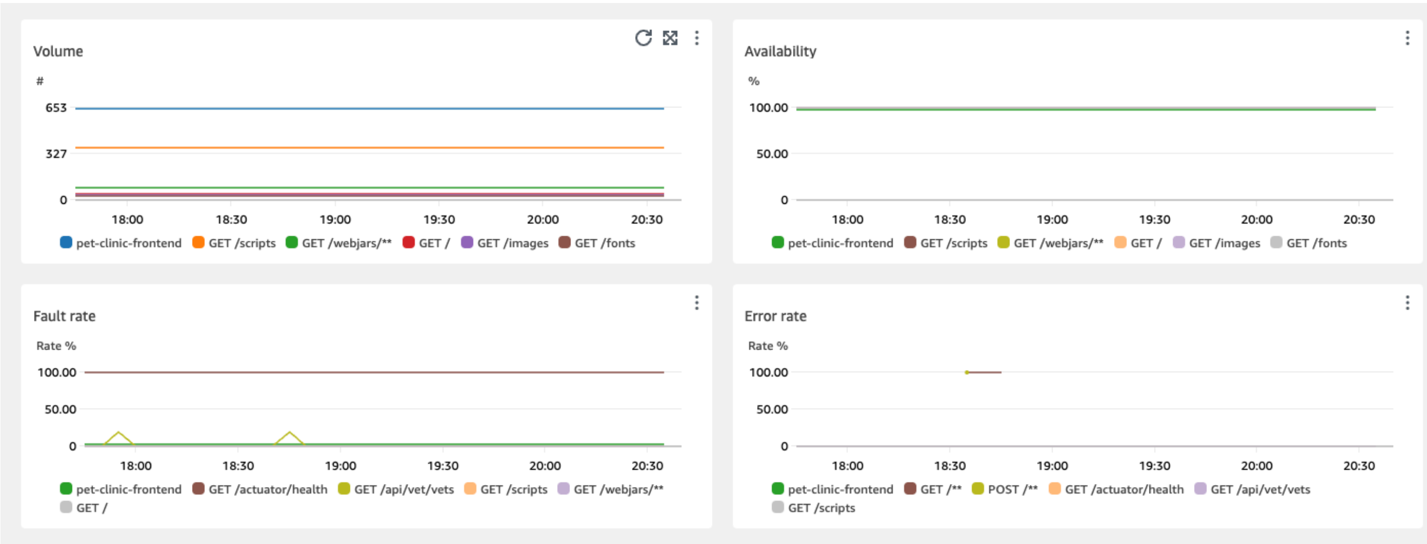


The **Overview** tab also displays a graph of dependencies with the highest latency across all services. Use the **p99**, **p90** and **p50** latency metrics to quickly assess which dependencies are contributing to your total service latency, as follows:



For example, the previous graph shows that 99% of the requests made to the **customer-service** dependency were completed in approximately 4,950 milliseconds. The other dependencies took less time.

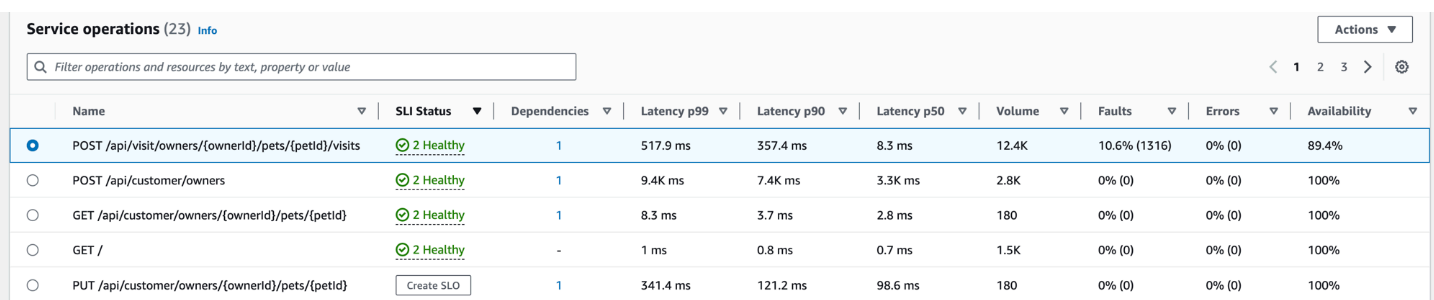
Graphs displaying the top four service operations by latency show the volume of requests, availability, fault rate, and error rate for those services, as shown in the following image:



The **Service details** section displays the details of the service including the **Account ID** and **Account label**.

View your service operations

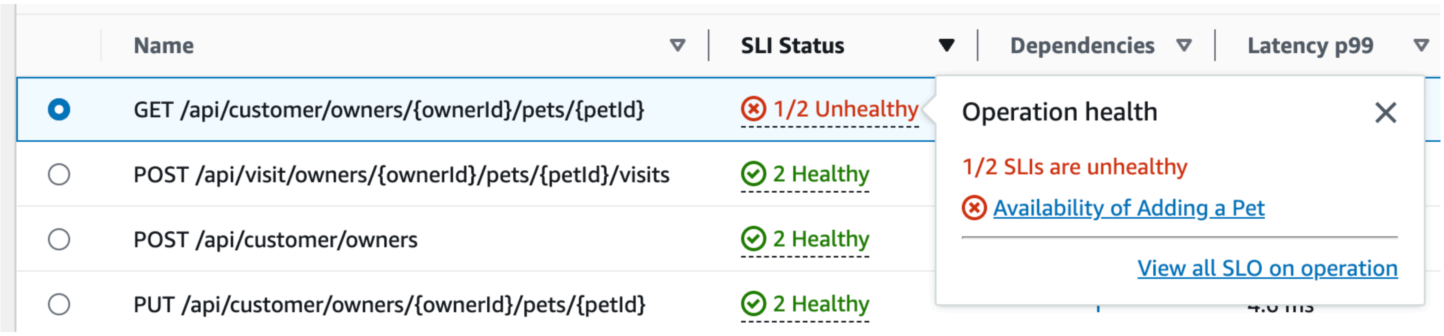
When you instrument your application, Application Signals discovers all of the service operations that your application calls. Use the **Service operations** tab to see a table that contains the service operations and a set of metrics that measure the performance of a selected operation. These metrics include SLI status, number of dependencies, latency, volume, faults, errors, and availability, as shown in the following image:



Name	SLI Status	Dependencies	Latency p99	Latency p90	Latency p50	Volume	Faults	Errors	Availability
POST /api/visit/owners/{ownerId}/pets/{petId}/visits	2 Healthy	1	517.9 ms	357.4 ms	8.3 ms	12.4K	10.6% (1316)	0% (0)	89.4%
POST /api/customer/owners	2 Healthy	1	9.4K ms	7.4K ms	3.3K ms	2.8K	0% (0)	0% (0)	100%
GET /api/customer/owners/{ownerId}/pets/{petId}	2 Healthy	1	8.3 ms	3.7 ms	2.8 ms	180	0% (0)	0% (0)	100%
GET /	2 Healthy	-	1 ms	0.8 ms	0.7 ms	1.5K	0% (0)	0% (0)	100%
PUT /api/customer/owners/{ownerId}/pets/{petId}	Create SLO	1	341.4 ms	121.2 ms	98.6 ms	180	0% (0)	0% (0)	100%

Filter the table to make it easier to find a service operation by choosing one or more properties from the filter text box. As you choose each property, you are guided through filter criteria and will see the complete filter below the filter text box. Choose **Clear filters** at any time to remove the table filter.

Choose the SLI status for an operation to display a popup containing a link to any unhealthy SLI, and a link to see all SLOs for the operation, as shown in the following table:



The service operations table lists the SLI status, the number of healthy or unhealthy SLIs, and the total number of SLOs for each operation.

Use SLIs to monitor latency, availability, and other operational metrics that measure the operational health of a service. Use an SLO to check the performance and health status of your services and operations.

To create an SLO, do the following:

- If an operation does not have an SLO, choose the **Create SLO** button within the **SLI Status** column.
- If an operation already has an SLO, do the following:
  - Select the radio button next to the operation name.
  - Choose **Create SLO** from the **Actions** down arrow at the top right of the table.

For more information, see service level objectives (SLOs).

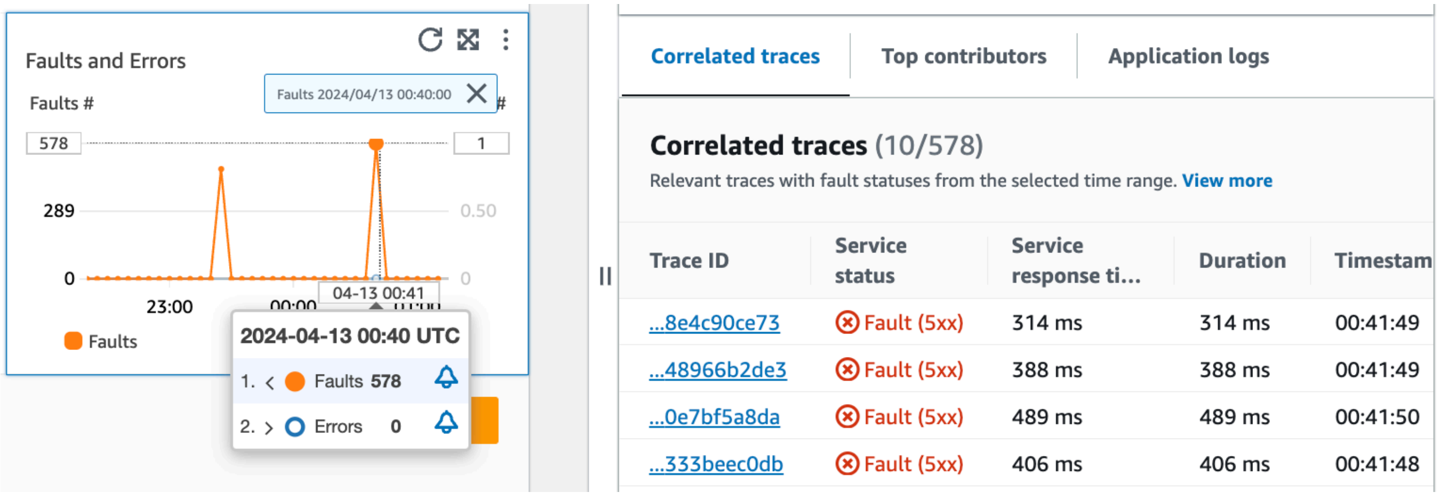
The **Dependencies** column shows the number of dependencies this operation calls. Choose this number to open the **Dependencies** tab filtered to the selected operation.

View service operations metrics, correlated traces, and application logs

Application Signals correlates service operation metrics with Amazon X-Ray traces, CloudWatch Container Insights, and application logs. Use these metrics to troubleshoot operational health issues. To view metrics as graphical information, do the following:

1. Select a service operation in the **Service operations** table to see a set of graphs for the selected operation above the table with metrics for **Volume and Availability**, **Latency**, and **Faults and Errors**.
2. Hover over a point in a graph to view more information.
3. Select a point to open a diagnostic pane that shows correlated traces, metrics, and application logs for the selected point in the graph.

The following image shows the tooltip that appears after hovering over a point in the graph, and the diagnostic pane which appears after clicking on a point. The tooltip contains information about the associated data point in the **Faults and Errors** graph. The pane contains **Correlated traces**, **Top contributors**, and **Application logs** associated with the selected point.



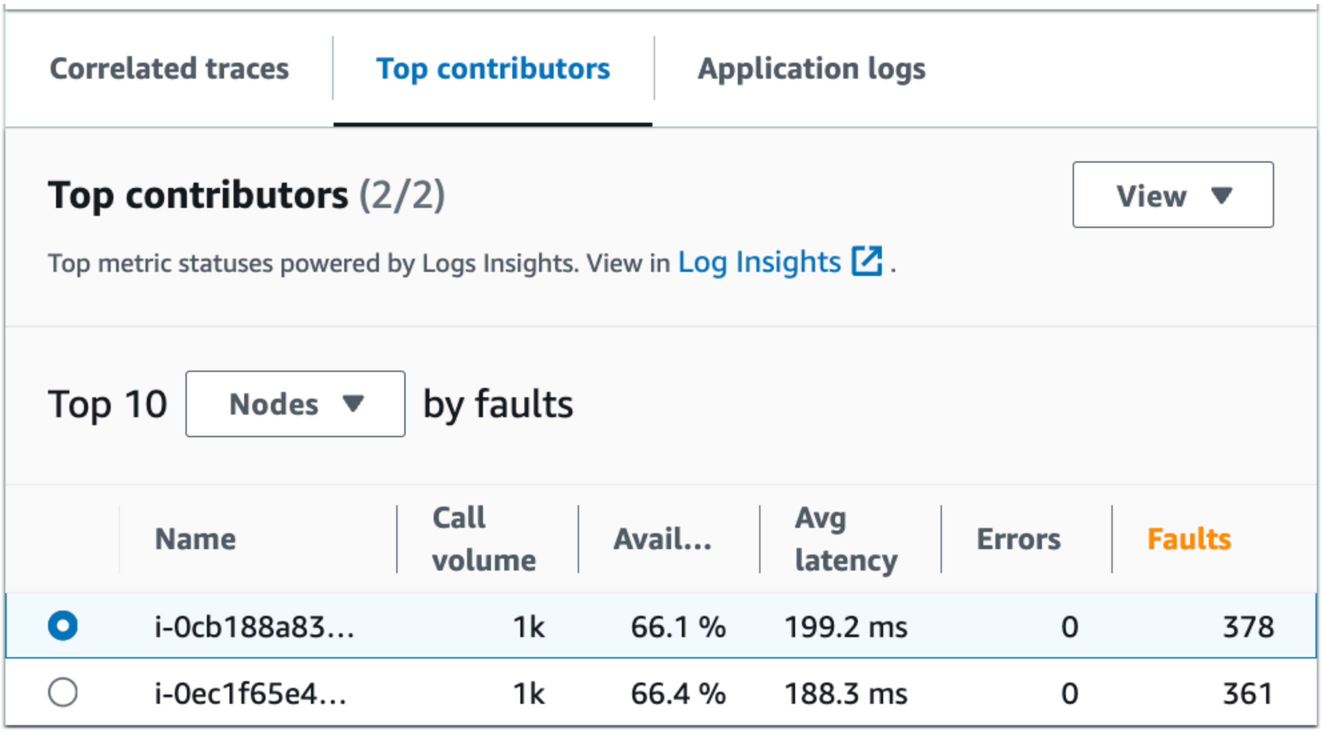
Correlated traces

Look at related traces to understand an underlying issue with a trace. You can check to see if correlated traces or any service nodes associated with them behave similarly. To examine correlated traces, choose a **Trace ID** from the **Correlated traces** table to open the X-Ray trace details page for the chosen trace. The trace details page contains a map of service nodes that are associated with the selected trace and a timeline of trace segments.

Top contributors

View the top contributors to find main input sources to a metric. Group contributors by different components to look for similarities within the group and understand how trace behavior differs between them.

The **Top contributors** tab gives metrics for **Call volume**, **Availability**, **Avg latency**, **Errors**, and **Faults** for each group. The following example image shows top contributors to a suite of metrics for an application deployed on an Amazon EKS platform:



The top contributors contains the following metrics:

- **Call volume** - Use the call volume to understand the number of requests per time interval for a group.
- **Availability** - Use availability to see what percentage of time that no faults were detected for a group.
- **Avg latency** - Use latency to check the average time that requests ran for a group over a time interval that depends on how long ago the requests that you are investigating were made. Requests that were made less than 15 days prior are evaluated over 1 minute intervals. Requests that were made between 15 and 30 days prior, inclusive, are evaluated over 5 minute intervals. For example, if you are investigating requests that caused a fault 15 days ago, the call volume metric is equal to the number of requests per 5 minute interval.
- **Errors** - The number of errors per group measured over a time interval.
- **Faults** - The number of faults per group over a time interval.

Top contributors using Amazon EKS or Kubernetes

Use information about the top contributors for applications deployed on Amazon EKS or Kubernetesto see operational health metrics grouped by **Node**, **Pod** and **PodTemplateHash**. The following definitions apply:

- A **pod** is a group of one or more Docker containers that share storage and resources. A pod is the smallest unit that can be deployed on a Kubernetes platform. Group by pods to check if errors are related to pod-specific limitations.
- A **node** is a server that runs pods. Group by nodes to check if errors are related to node-specific limitations.
- A **pod template hash** is used to find a particular version of a deployment. Group by pod template hash to check if errors are related to a particular deployment.

Top contributors using Amazon EC2

Use information about the top contributors for applications deployed on Amazon EKS to see operational health metrics grouped by instance ID, and auto scaling group. The following definitions apply:

- An **Instance ID** is a unique identifier for the Amazon EC2 instance that your service runs. Group by instance ID to check if errors are related to a specific Amazon EC2 instance.
- An auto scaling group is a collection of Amazon EC2 instances that allow you to scale up or down the resources you need to serve your application requests. Group by auto scaling group if you want to check if errors are limited in scope to the instances inside the group.

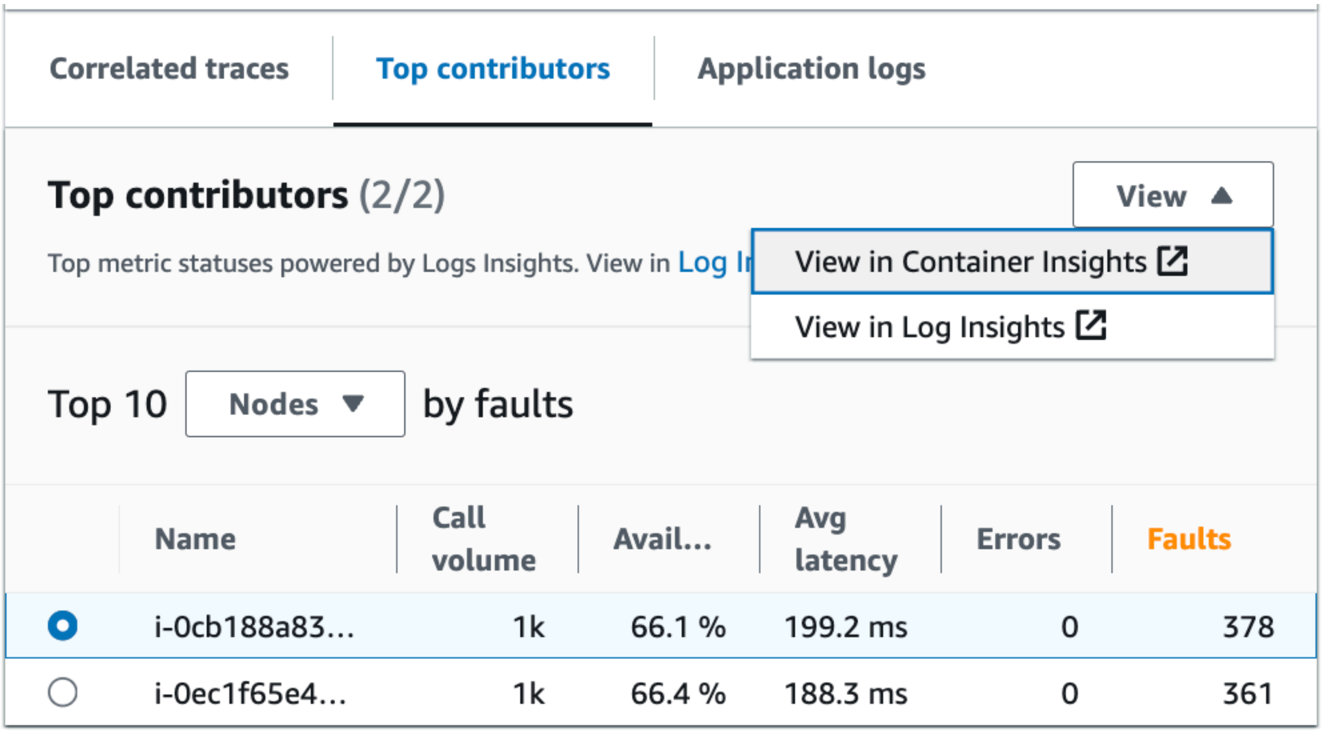
Top contributors using a custom platform

Use information about the top contributors for applications deployed using custom instrumentation to see operational health metrics grouped by **Host name**. The following definitions apply:

- A host name identifies a device such as an endpoint or Amazon EC2 instance that is connected to a network. Group by host name to check if your errors are related to a specific physical or virtual device.

View top contributors in Log Insights and Container Insights

View and modify the automatic query that generated metrics for your top contributors in Log Insights. View infrastructure performance metrics by specific groups such as pods or nodes in Container Insights. You can sort clusters, nodes or workloads by resource consumption and quickly identify anomalies or and mitigate risks pro-actively before end user experience is impacted. An image showing how to select these options follows:



In **Container Insights**, you can view metrics for your Amazon EKS or Amazon ECS container that are specific to the grouping of your top contributors. For example, if you grouped by pod for an EKS container to generate top contributors, container insights will show metrics and statistics filtered for your pod.

In **Log Insights**, you can modify the query that generated the metrics under **Top contributors** using the following steps:

1. Select **View in Log Insights**. The **Logs Insights** page that opens contains an query that is automatically generated and contains the following information:

   - The log cluster group name.
   - The operation that you were investigating with CloudWatch.
   - The aggregate of the operational health metric interacted with on the graph.

The log results are automatically filtered to show data from the last five minutes before you selected the data point on the service graph.

2. To edit the query, replace the generated text with your changes. You can also use the **Query generator** to help you generate a new query, or update the existing query.

Application logs

Use the query in the **Application logs** tab to generate logged information for your current log group, service and insert a timestamp. A log group is a group of log streams that you can define when you configure your application.

Use a log group to organize logs with similar characteristics including the following:

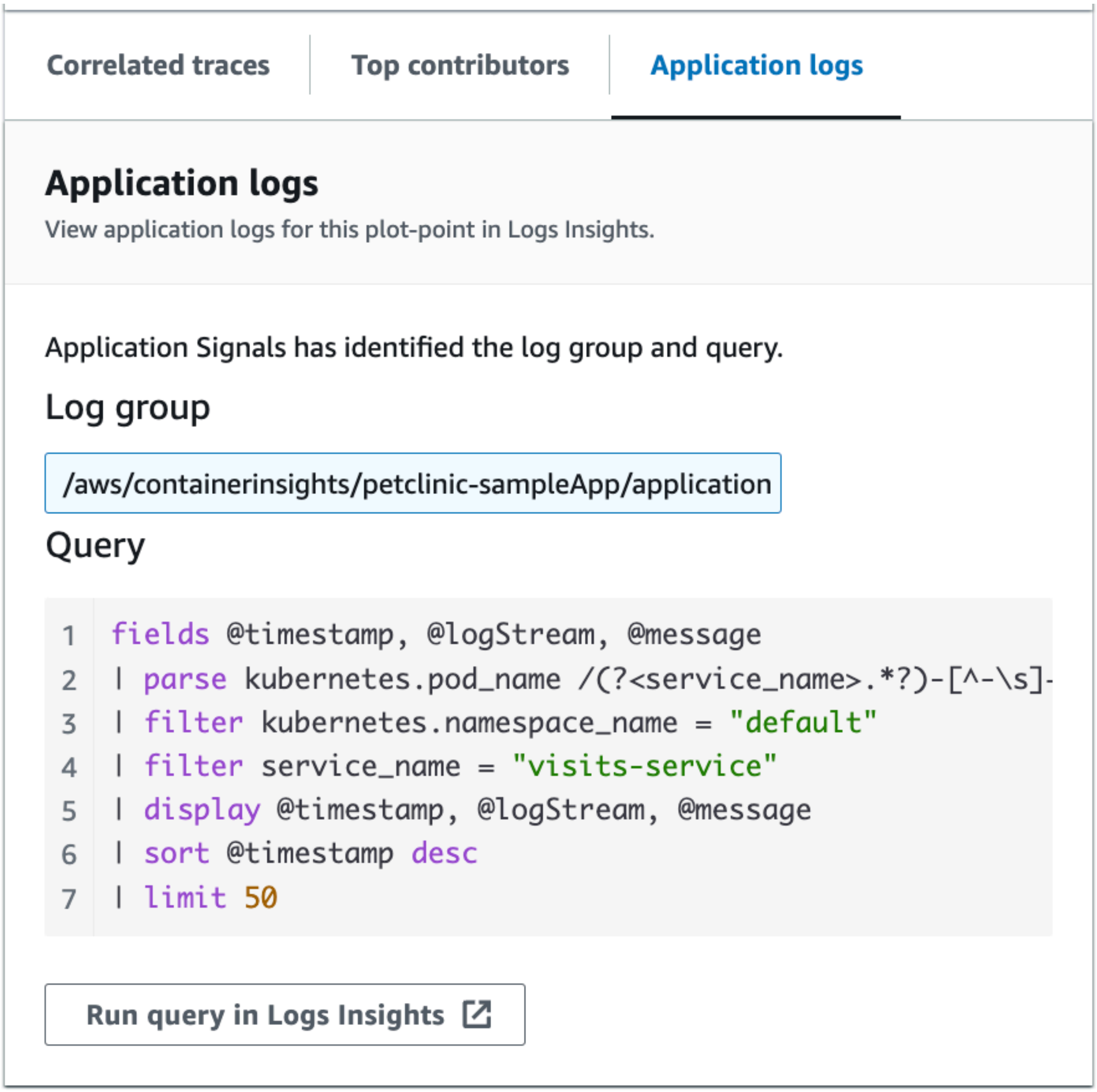
- Capture logs from a specific organization, source or function.
- Capture logs that are accessed by a particular user.
- Capture logs for a specific time period.

Use these log streams to track specific groups or time frames. You can also set up monitoring rules, alarms and notifications for these log groups. For more information about log groups, see Working with log groups and log streams.

The application logs query returns the logs, recurring text patterns and graphical visualizations for your log groups.

To run the query, select **Run query in Logs Insights** to either run the automatically generated query or modify the query. To edit the query, replace the automatically generated text with your changes. You can also use the **Query generator** to help you generate a new query or update the existing query.

The following image shows the sample query that is automatically generated based on the selected point in the service operations graph:



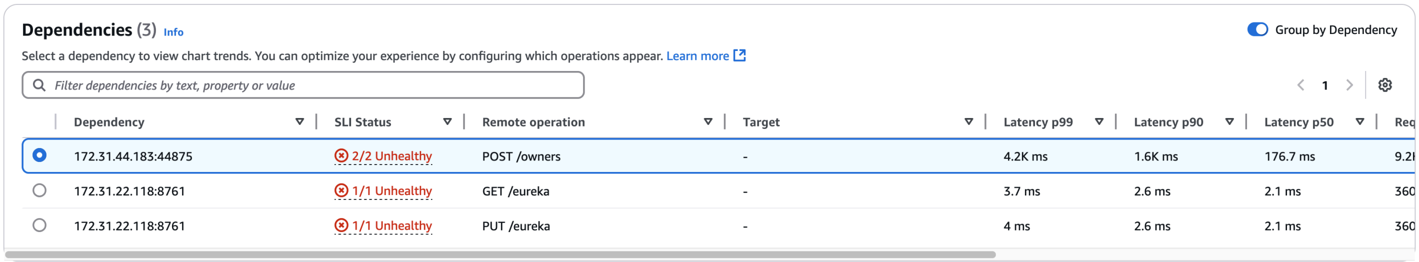
In the preceding image, CloudWatch has automatically detected the log group that is associated with your selected point, and included it in a generated query.

View your service dependencies

Choose the **Dependencies** tab to display the **Dependencies** table and a set of metrics for the dependencies of all service operations or a single operation. The table contains a list of dependencies discovered by Application Signals, including metrics for SLI status, latency, call volume, fault rate, error rate, and availability.

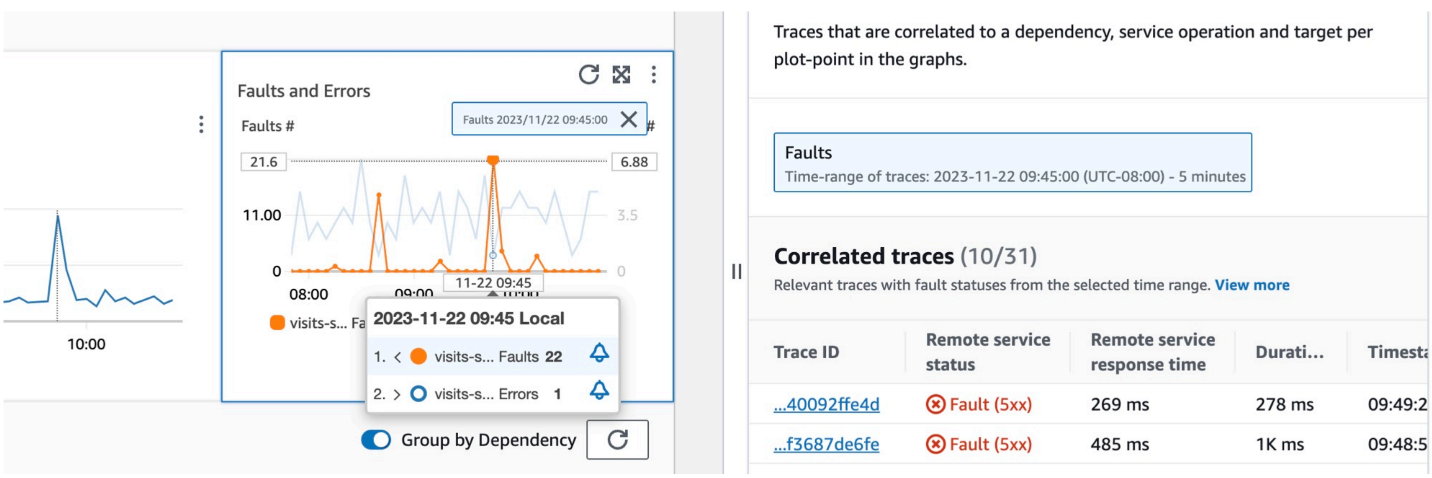
At the top of the page, choose an operation from the down arrow list to view its dependencies, or choose **All** to see dependencies for all operations.

Filter the table to make it easier to find what you're looking for, by choosing one or more properties from the filter text box. As you choose each property, you are guided through filter criteria and will see the complete filter below the filter text box. Choose **Clear filters** at any time to remove the table filter. Select **Group by Dependency** at the top right of the table to group dependencies by service and operation name. When grouping is turned on, expand or collapse a group of dependencies with the **+** icon next to the dependency name.



The **Dependency** column displays the dependency service name, while the **Remote Operation** column displays the service operation name. The **SLI status** column displays the number of healthy or unhealthy SLIs along with the total number of SLIs for each dependency. When calling Amazon services, the **Target** column displays the Amazon resource, such as DynamoDB table or Amazon SNS queue.

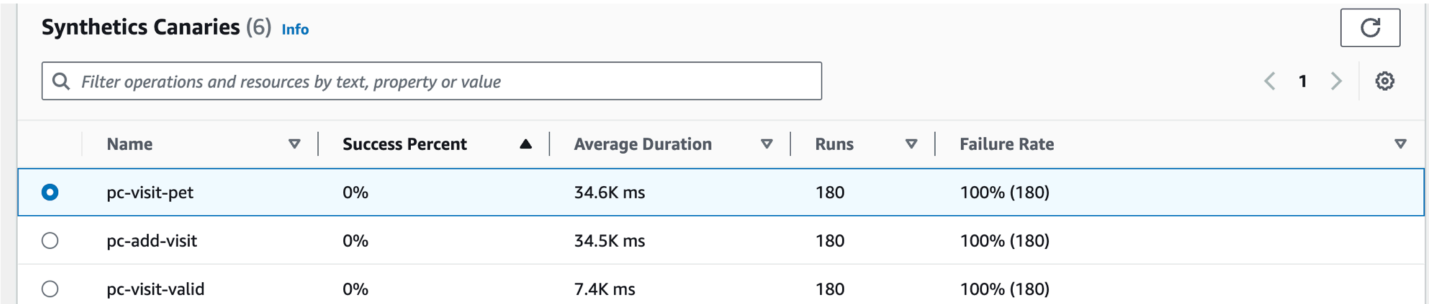
To select a dependency, select the option next to a dependency in the **Dependencies** table. This shows a set of graphs that display detailed metrics for call volume, availability, faults, and errors. Hover over a point in a graph to see a popup containing more information. Select a point in a graph to open a diagnostic pane that shows correlated traces for the selected point in the graph. Choose a trace ID from the **Correlated traces** table to open the [X-Ray Trace details](#) page for the selected trace.



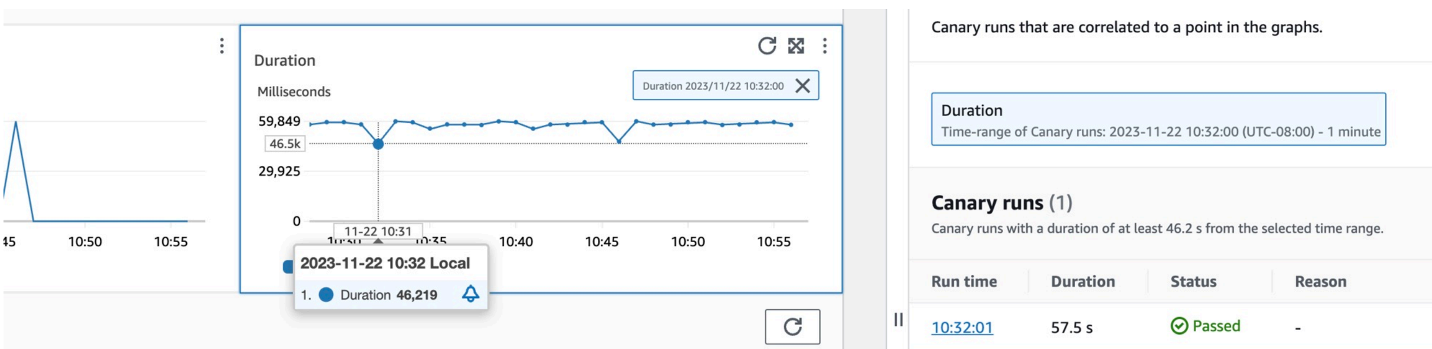
View your Synthetics canaries

Choose the **Synthetics Canaries** tab to display the **Synthetics Canaries** table, and a set of metrics for each canary in the table. The table includes metrics for success percentage, average duration, runs, and failure rate. Only canaries that are enabled for Amazon X-Ray tracing are displayed.

Use the filter text box in the synthetics canaries table to find the canary that you are interested in. Each filter that you create appears below the filter text box. Choose **Clear filters** at any time to remove the table filter.



Select the radio button next to the name of the canary to see a set of tabs containing graphs detailed metrics including success percentage, errors and duration. Hover over a point in a graph to see a popup containing more information. Select a point in a graph to open a diagnostic pane that shows canary runs that correlate to the selected point. Select a canary run and choose the **Run time** to see artifacts for your selected canary run including logs, HTTP Archive (HAR) files, screenshots, and suggested steps to help you troubleshoot problems. Choose **Larn more** to open the CloudWatch Synthetics Canaries page next to **Canary runs**.

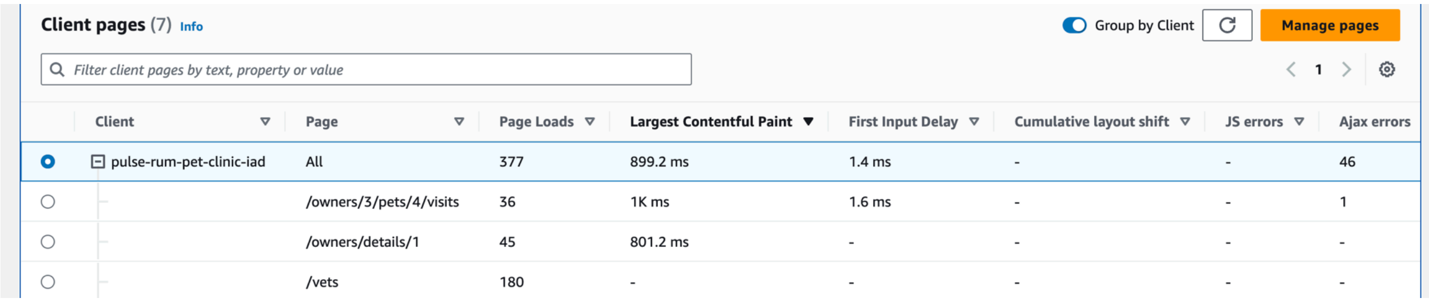


View your client pages

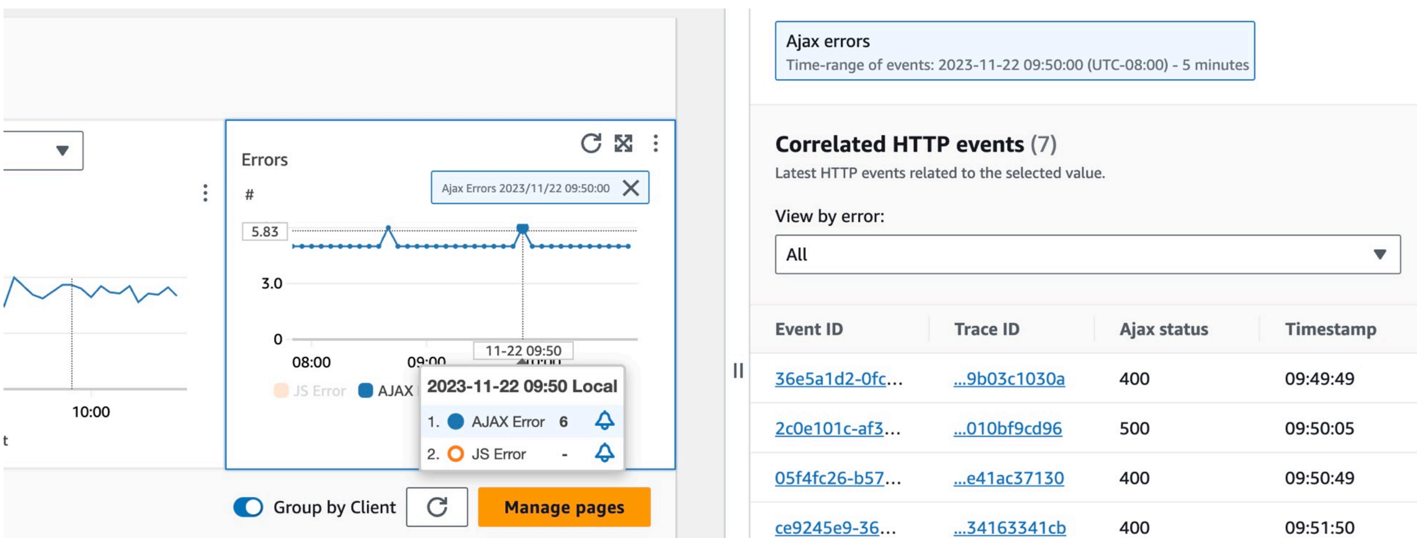
Choose the **Client pages** tab to display a list of client web pages that call your service. Use the set of metrics for the selected client page to measure the quality of your client's experience when interacting with a service or application. These metrics include page loads, web vitals, and errors.

To display your client pages in the table, you must configure your CloudWatch RUM web client for X-Ray tracing and turn on Application Signals metrics for your client pages. Choose **Manage pages** to select which pages are enabled for Application Signals metrics.

Use the filter text box to find the client page or application monitor that you are interested in below the filter text box. Choose **Clear filters** to remove the table filter. Select **Group by Client** to group client pages by client. When grouped, choose the **+** icon next to a client name to expand the row and see all pages for that client.



To select a client page, select the option next to a client page in the **Client pages** table. You will see a set of graphs that display detailed metrics. Hover over a point in a graph to see a popup containing more information. Select a point in a graph to open a diagnostic pane that shows correlated performance navigation events for the selected point in the graph. Choose an event ID from the list of navigation events to open the CloudWatch RUM Page view for the chosen event.



> **Note**
>
> To see AJAX errors within your client pages, use the CloudWatch RUM web client version 1.15 or newer.

Currently, up to 100 operations, canaries, and client pages, and up to 250 dependencies, can be shown per service.

View your application topology and monitor operational health with the CloudWatch service map

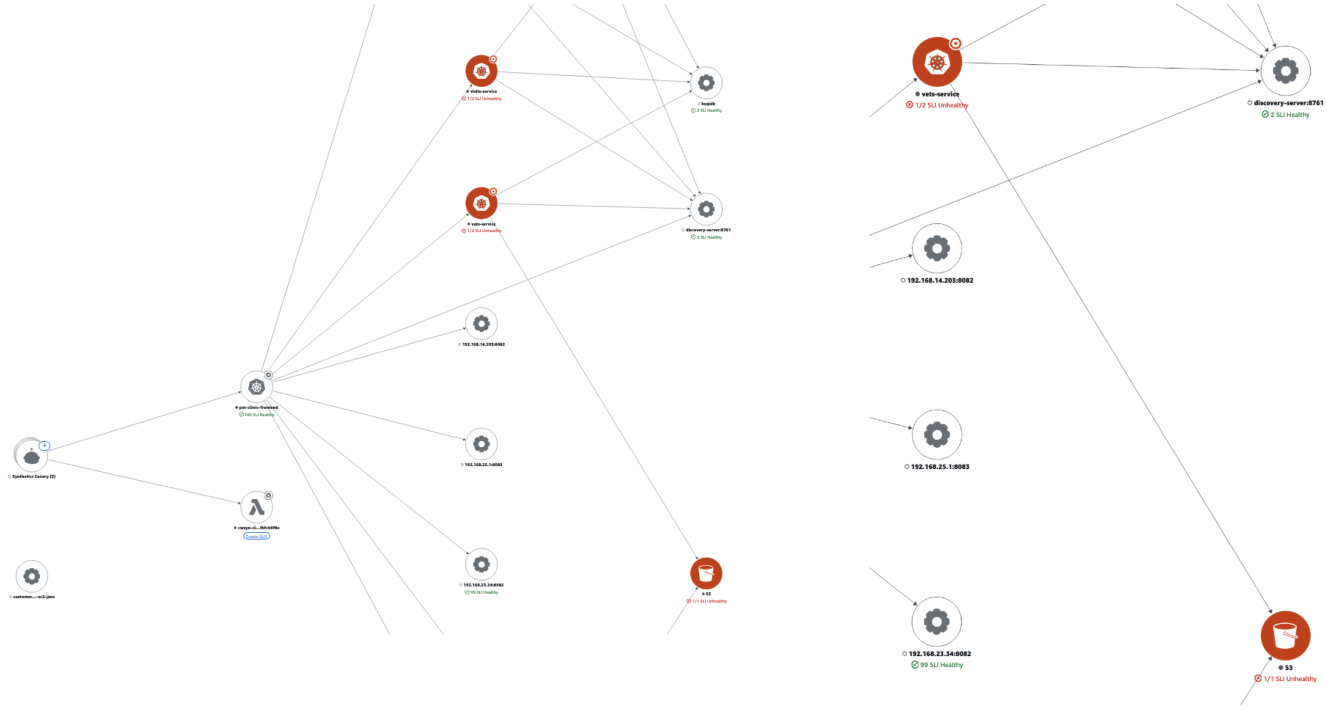
> **Note**
>
> The CloudWatch service map replaces the ServiceLens map. To see a map of your application based on Amazon X-Ray traces, open the X-Ray Trace Map. Choose **Trace Map** under the **X-Ray** section in the left navigation pane of the CloudWatch console.

Use the service map to view the topology of your application clients, synthetics canaries, services and dependencies, and monitor operational health. To view the service map, open the CloudWatch console and choose **Service Map** under the **Application Signals** section in the left navigation pane.

After you enable your application for Application Signals, use the service map to make it easier to monitor your application's operational health:

- View connections between client, canary, service, and dependency nodes to help you understand your application topology and execution flow. This is especially helpful if your service operators are not your development team.
- See which services are meeting or not meeting your service level objectives (SLOs). When a service is not meeting your SLOs, you can quickly identify whether a downstream service or dependency might be contributing to the issue or impacting multiple upstream services.
- Select an individual client, synthetics canary, service, or dependency node to see related metrics. The Service details page shows more detailed information about operations, dependencies, synthetics canaries, and client pages.
- Filter and zoom the service map to make it easier to focus on a part of your application topology, or see the entire map. Create a filter by choosing one or more properties from the filter text box. As you choose each property, you are guided through filter criteria. You will see the complete filter below the filter text box. Choose **Clear filters** at any time to remove the filter.

The following example service map shows services with edges connecting them to components that they interact with. If an SLO is defined, the service map also shows health status.

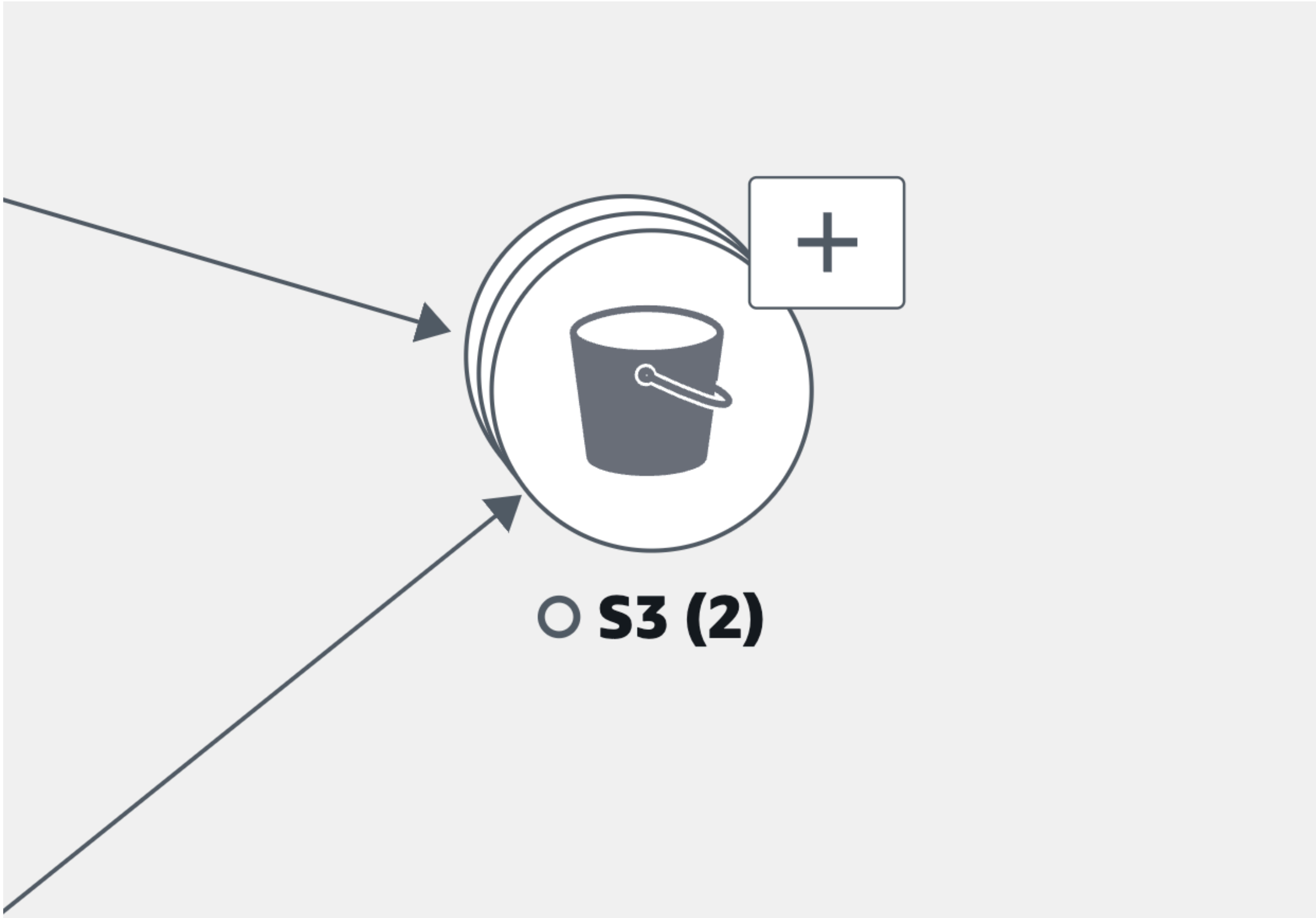


Explore the service map

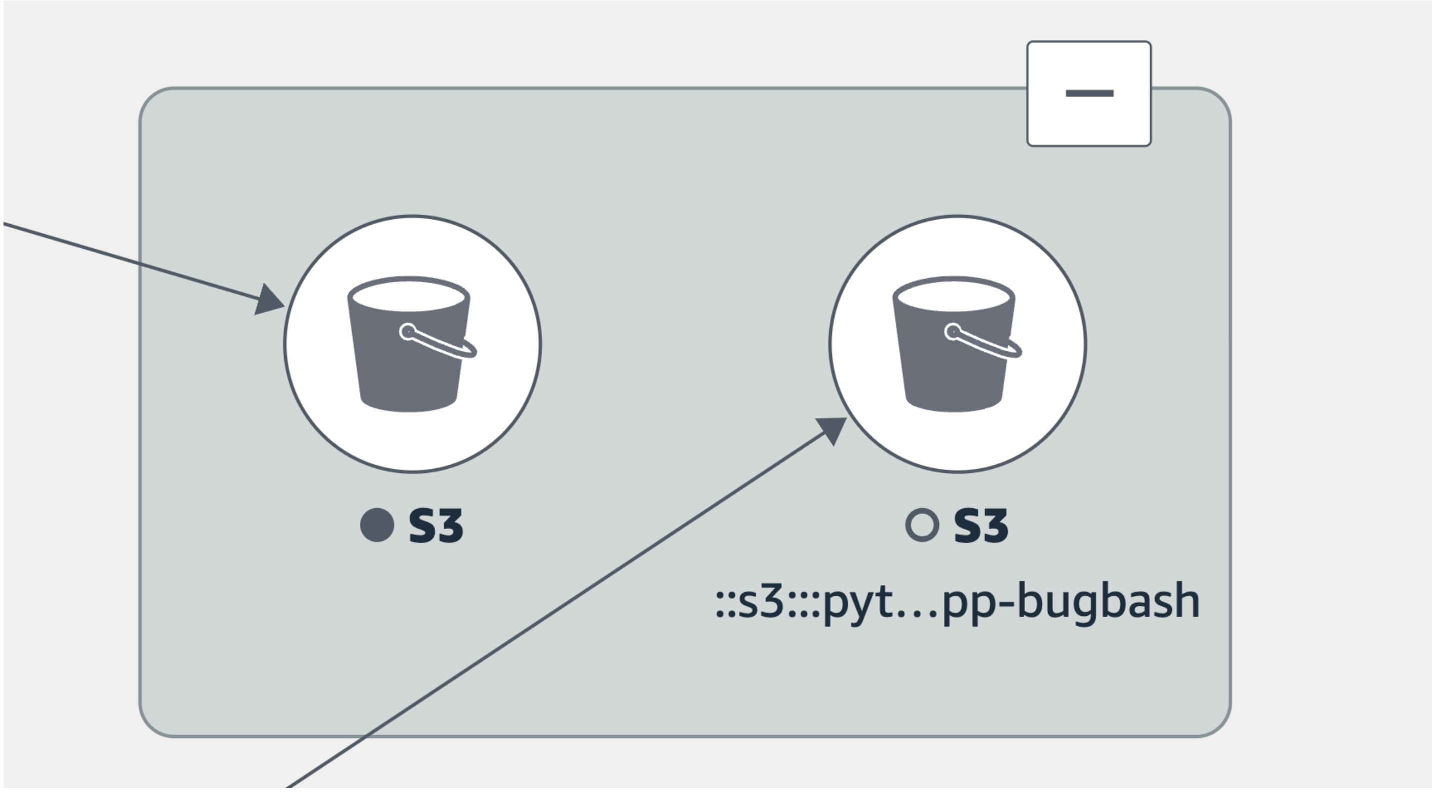
After you have enabled your application for Application Signals, the service map displays nodes representing your services and their dependencies.

Turn on active tracing for your CloudWatch RUM clients and synthetics canaries to see client and canary nodes on the map.

By default, canaries, RUM clients, and Amazon service dependencies of the same kind are grouped together into a single expandable icon in the service map. Service dependencies outside of Amazon are not grouped together by default. For example, in the following image, all Amazon S3 buckets are grouped together under one expandable icon:



In the previous image, the label between the Amazon S3 grouping and originating service displays the number of edges to the group in parenthesis under the dependency's icon. Select the (+) icon to expand the group and see its individual elements, as shown in the following image:

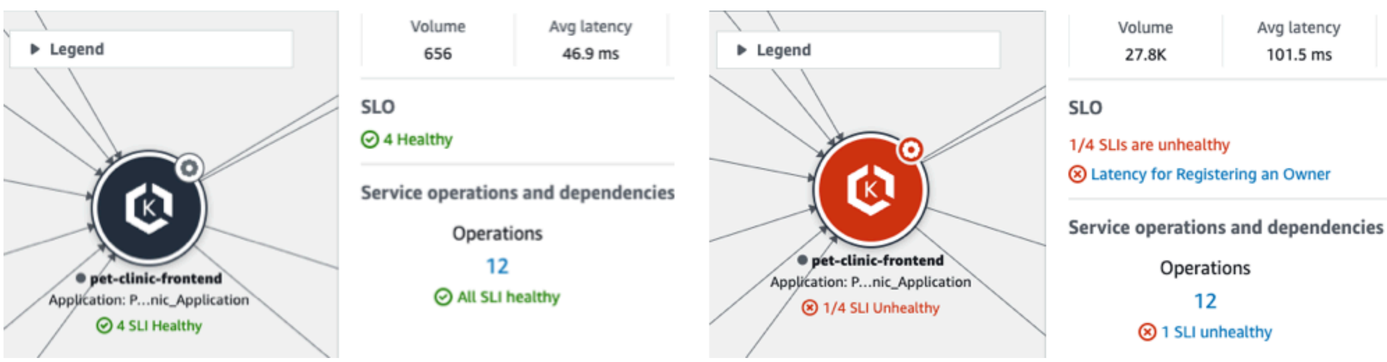


Choose a tab for information about exploring each kind of node and the edges (connections) between them.

View your application services

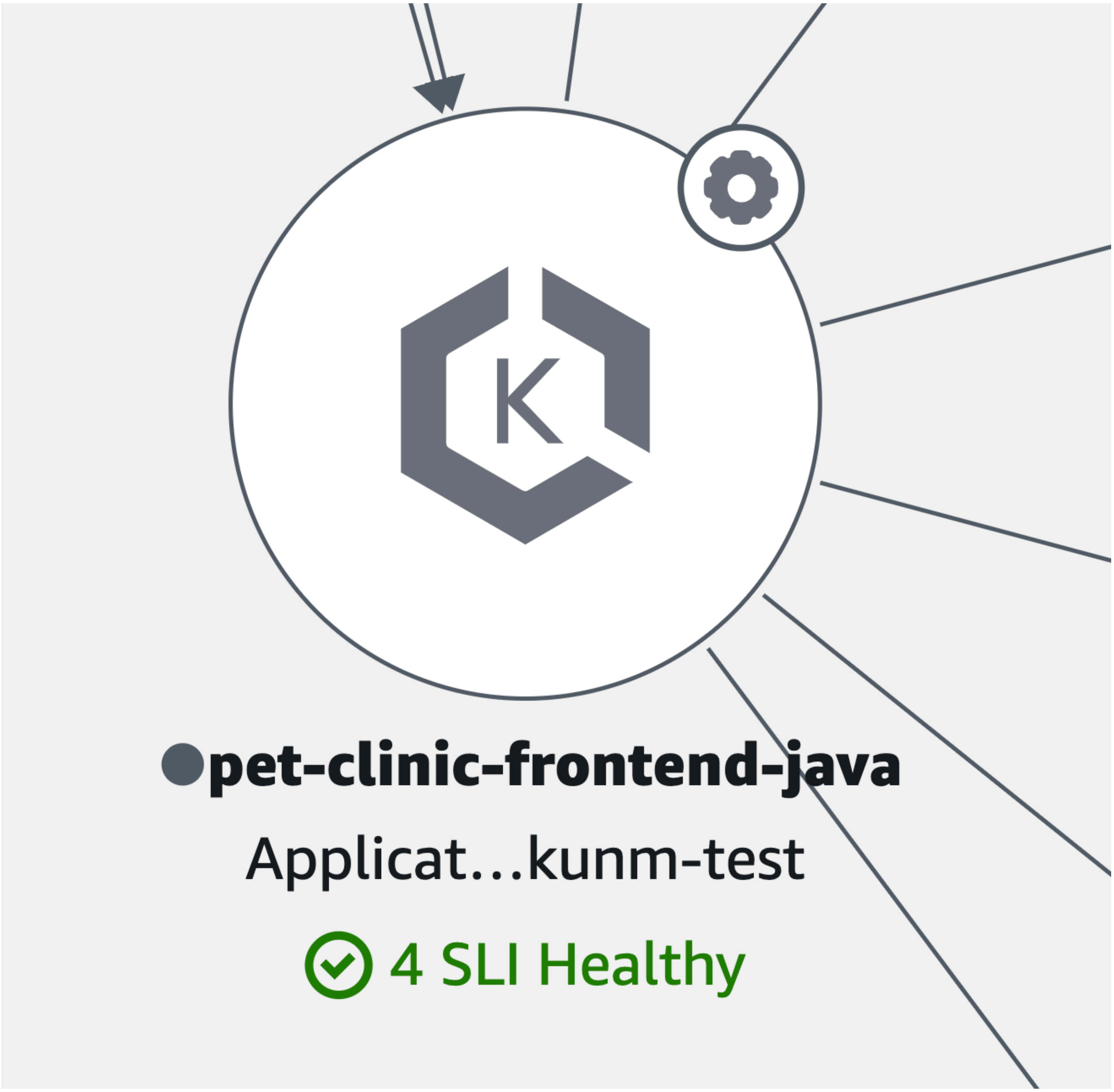
You can view your application services and the status of their SLOs and service level indicators (SLIs) in the **Service Map**. If you didn't create SLOs for a service, choose the **Create SLO** button below the service node.

The **Service Map** displays all of your services. It also shows the customers and canaries that consume the service and the dependencies that your services calls, as shown in the following image:



The following icons represent examples of application services in the service map:

- Amazon Elastic Kubernetes Service:



- A Kubernetes container:

- Amazon Elastic Compute Cloud (Amazon EC2):

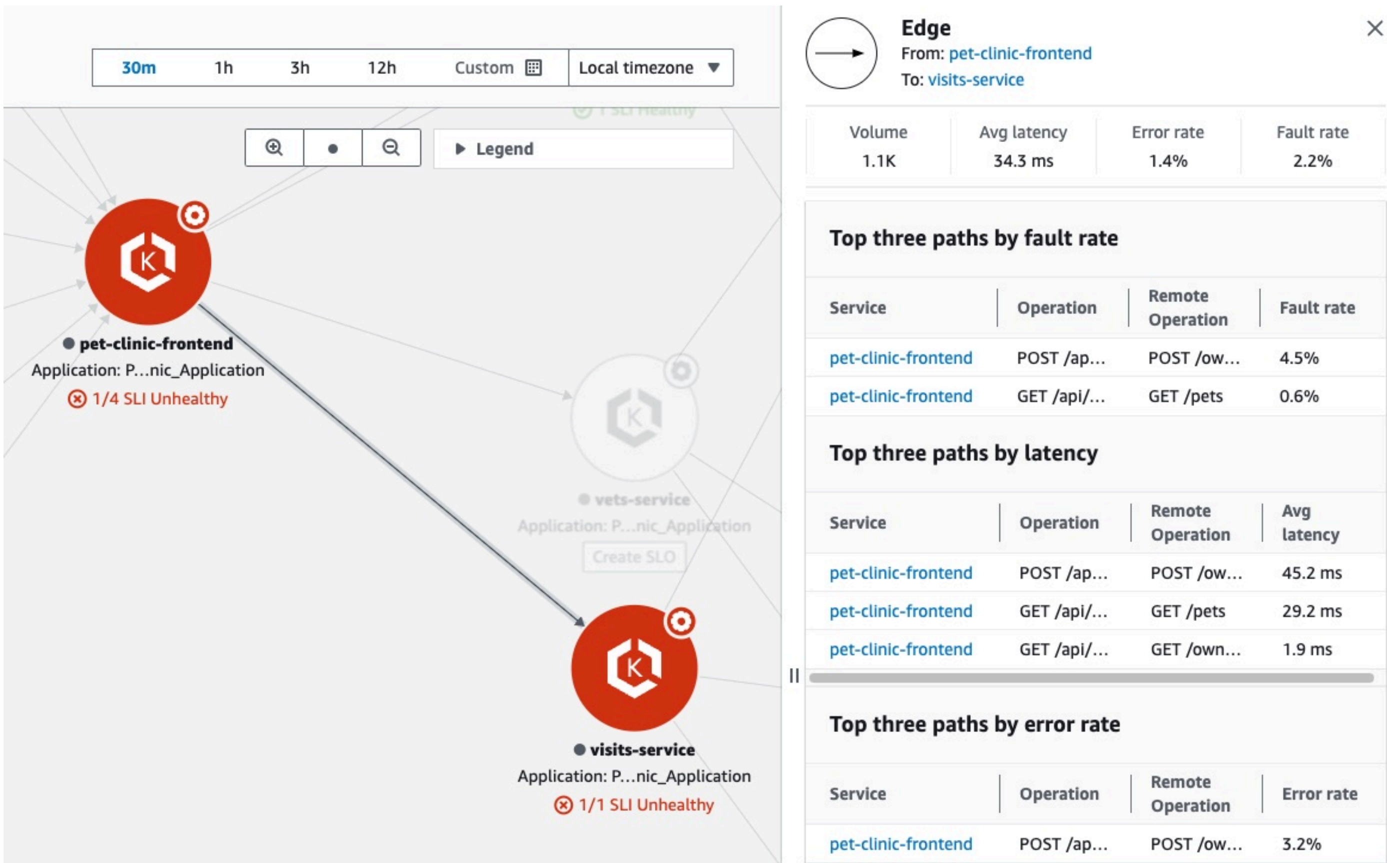
- Other application service types not previously listed:

When you select a service node, a pane opens displaying detailed service information:

- Metrics for call volume, latency, error, and fault rate.
- The number of SLIs and SLOs that are `healthy` or `unhealthy`.
- The option to view more information about an SLO.
- The number of service operations, dependencies, synthetics canaries, and client pages.
- The option to select each number to open the Service details page for it.
- The application name, if you have associated the underlying compute resource with an application using AppRegistry or the Applications card on the Amazon Web Services Management Console home page.

  - Choose the application name to display the application details in the myApplications console page.
- The `Cluster`, `Namespace`, and `Workload` for services hosted in Amazon EKS, or `Environment` for services hosted in Amazon ECS or Amazon EC2. For Amazon EKS-hosted services, choose any link to open CloudWatch Container Insights.

Select an edge or connection between a service node and a downstream service or dependency node. This opens a pane containing top paths by fault rate, latency, and error rate, as shown in the following example image. Choose any link in the pane to open the Service details page and see detailed information for the chosen service or dependency.

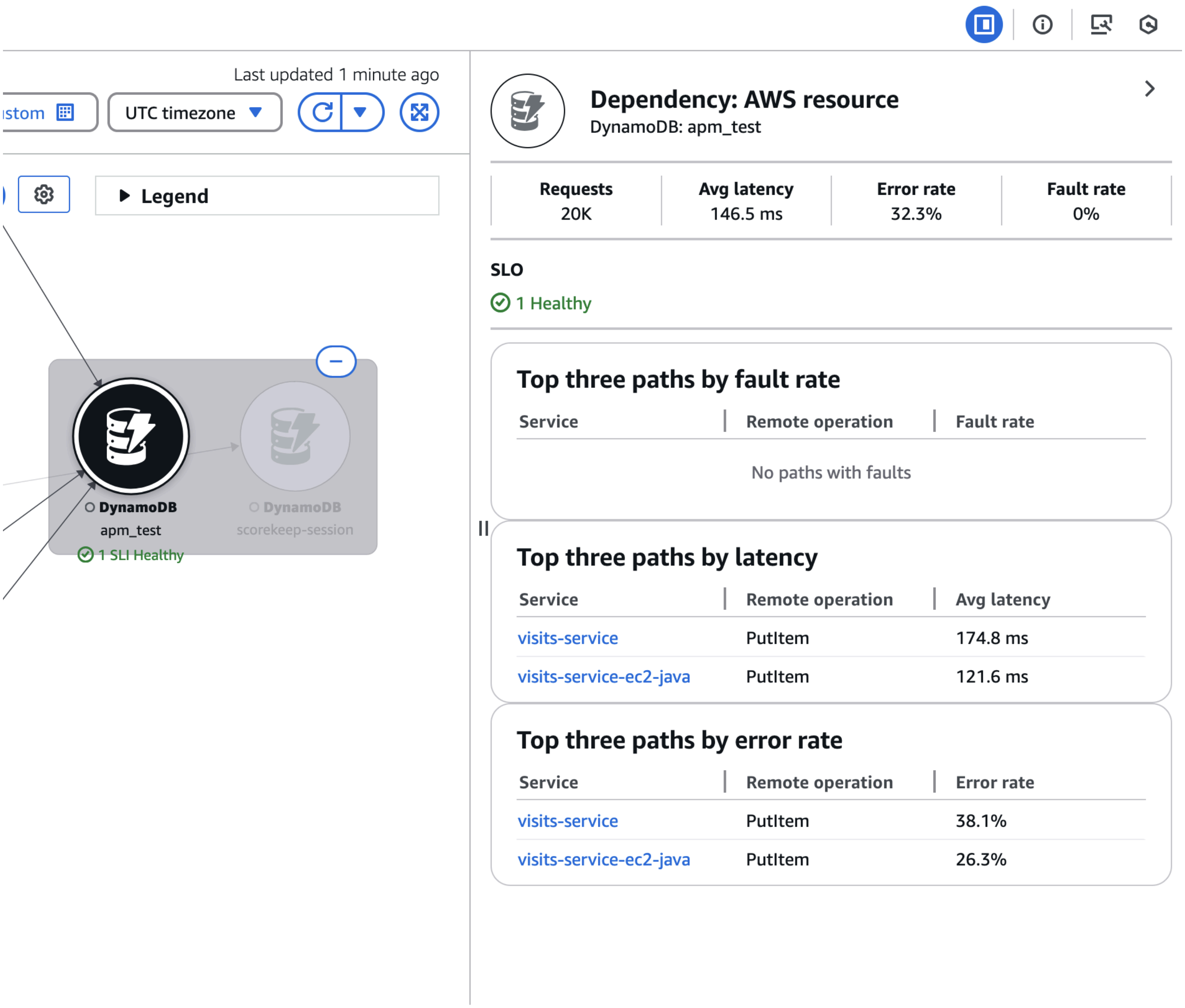


View dependencies

Your application dependencies are displayed on the service map, connected to the services that call them.

Choose a dependency node to open a pane containing top paths by fault rate, latency, and error rate. Choose any service or target link to open the Service Details page and see detailed information about the chosen service or dependency target, as shown in the example image below:

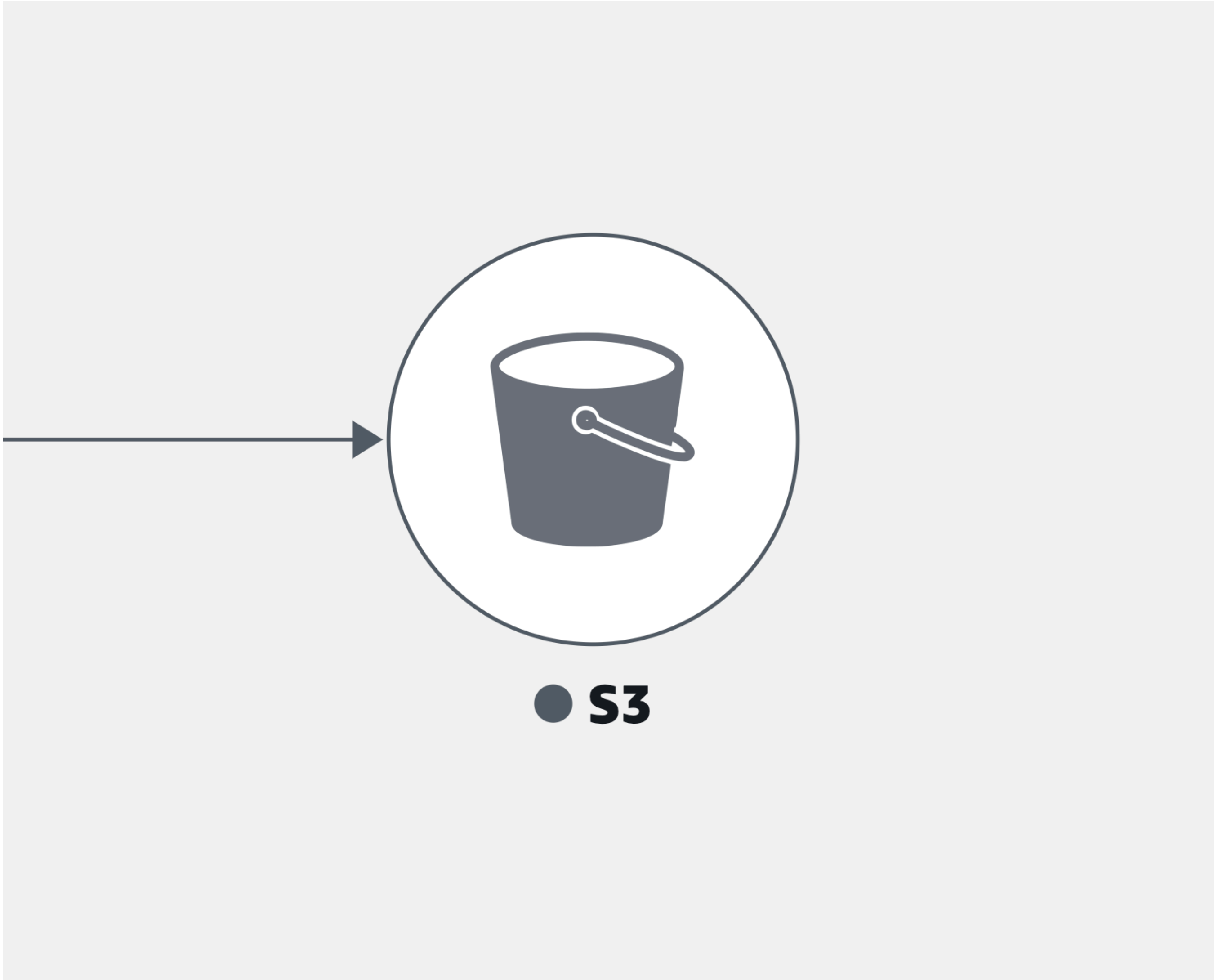
You can view the dependencies and the status of SLOs created on the dependencies.



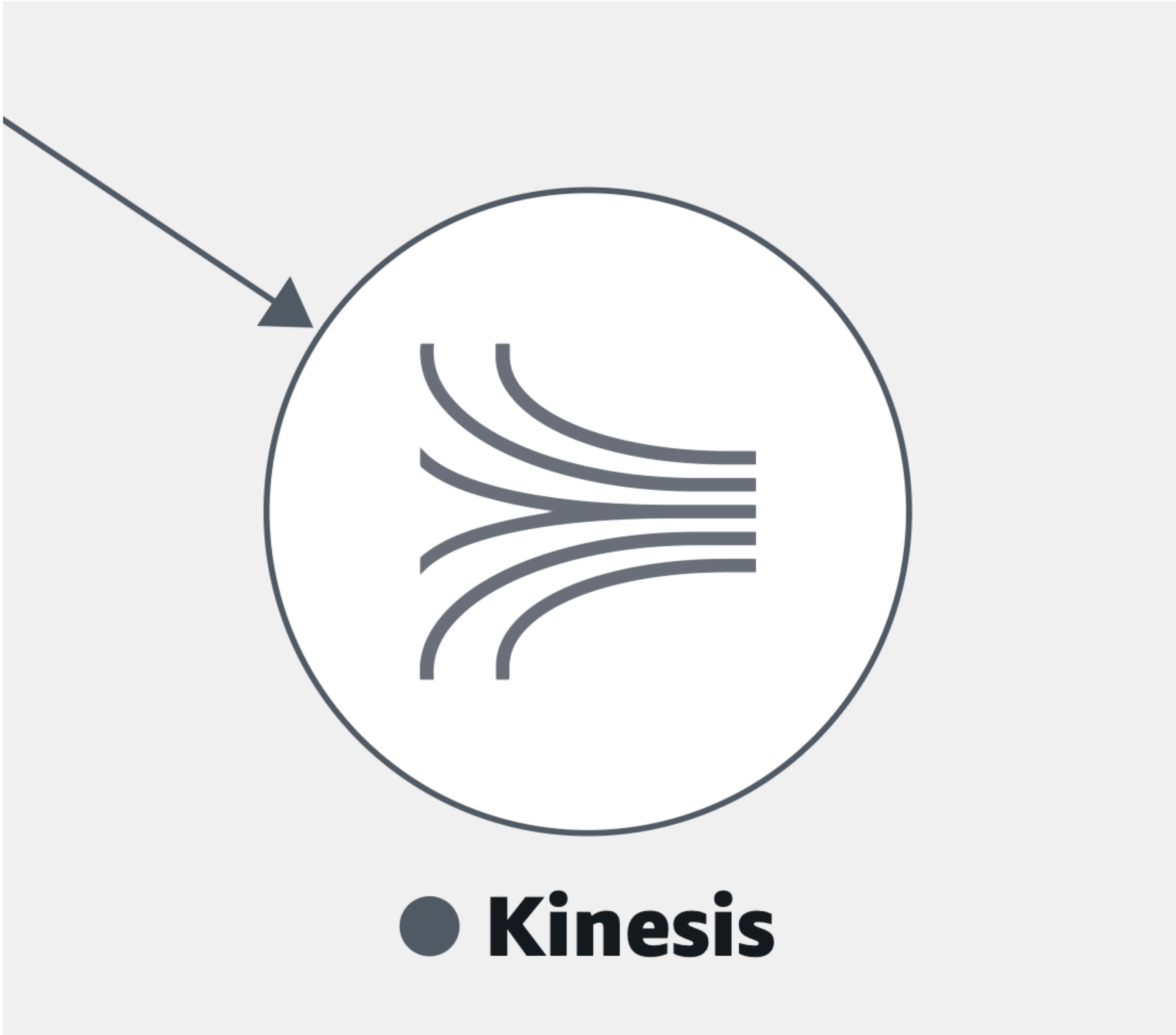
Service dependencies are grouped together by default into a single expandable icon. Select the (+) icon, as shown in the previous image, to expand the group and see its individual elements.

The following icons represent examples of dependency nodes in the service map:

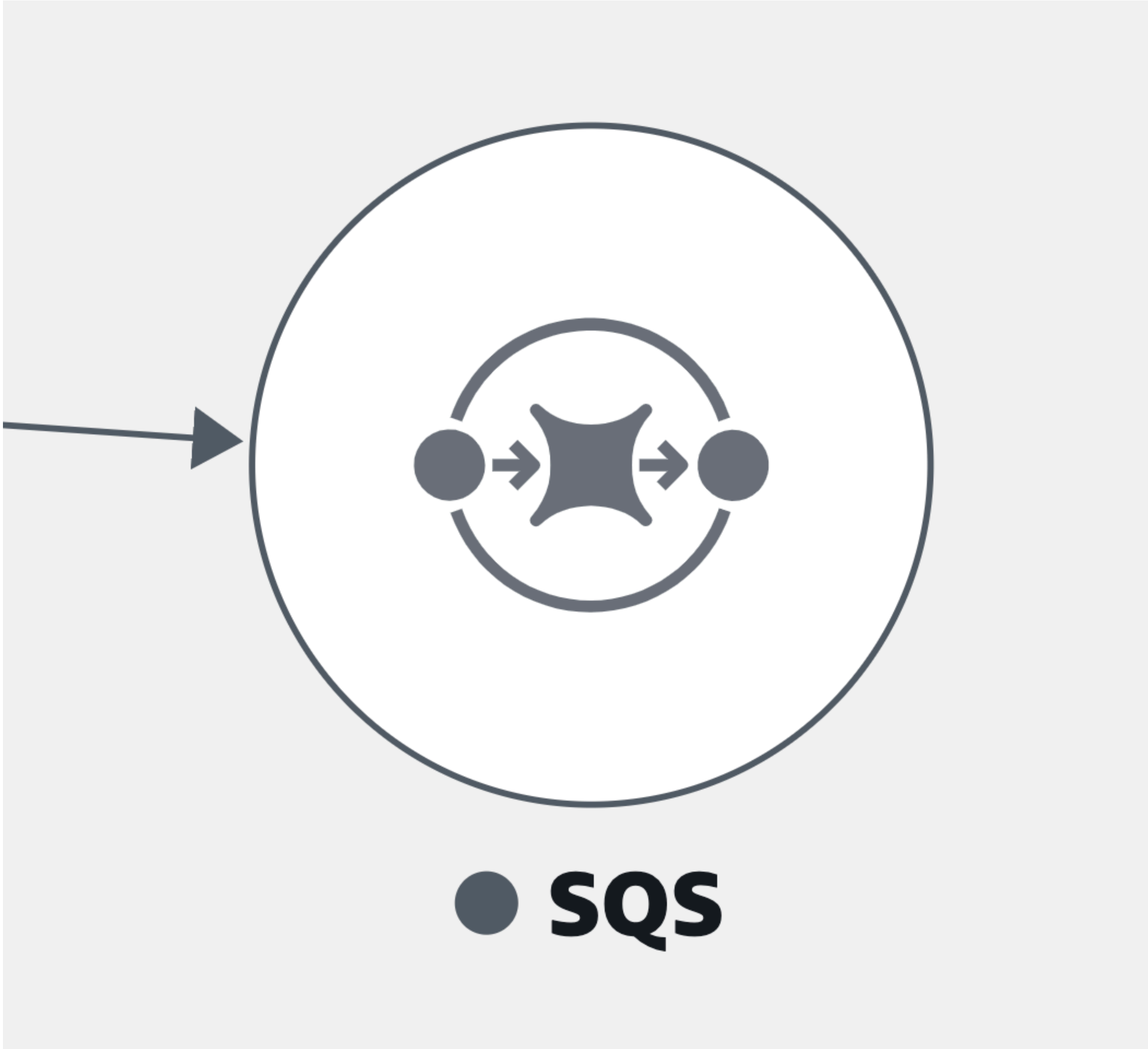
- An [Amazon S3](https://aws.amazon.com/s3/) bucket:



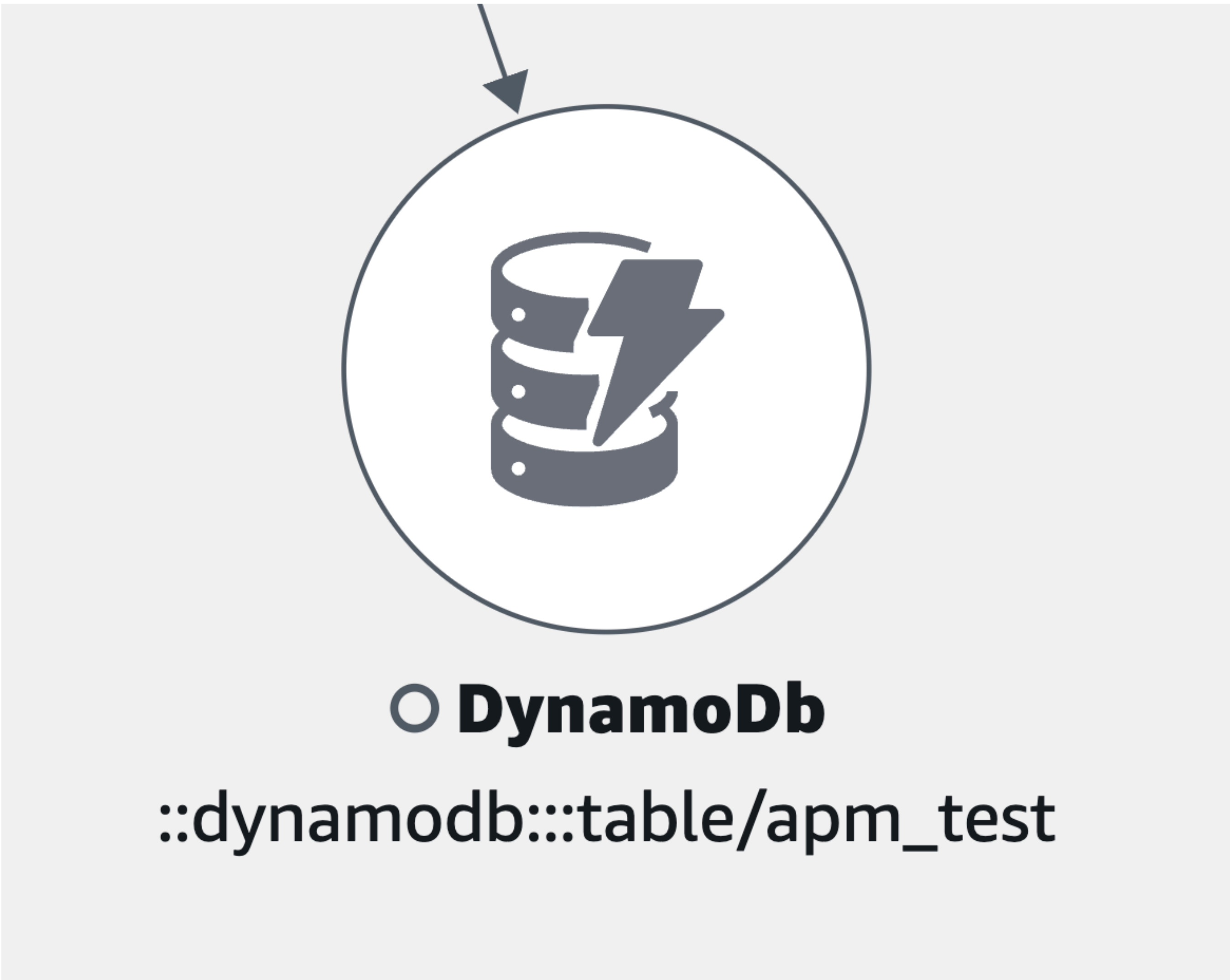
- An [Amazon Kinesis](https://docs.aws.amazon.com/kinesis/) stream:



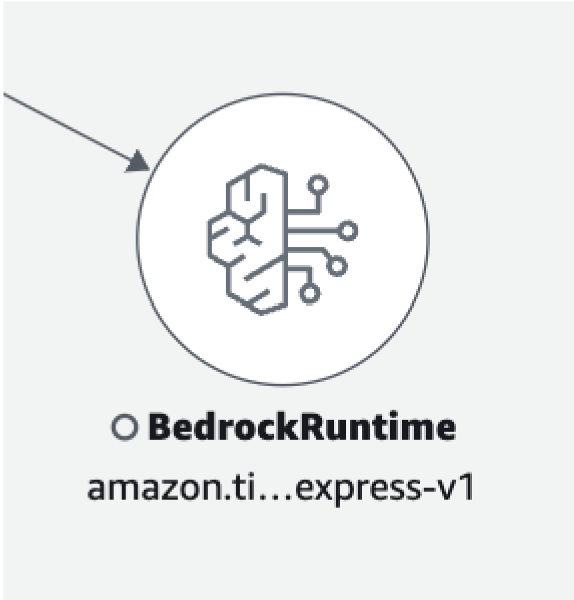
- Amazon Simple Queue Service (Amazon SQS):



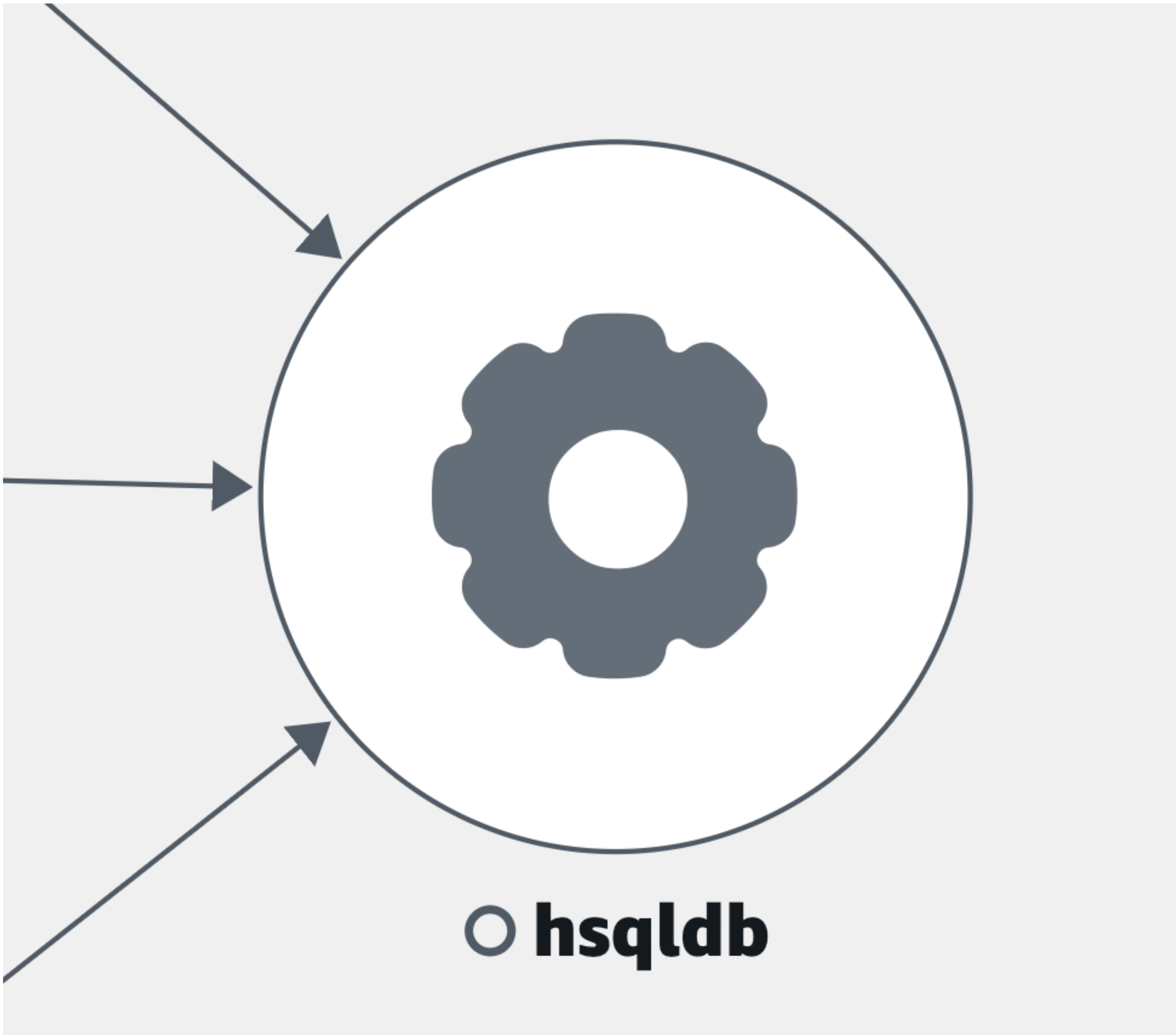
- An [Amazon DynamoDB](https://docs.aws.amazon.com/dynamodb/) table:



- An Amazon Bedrock model:



- Other dependency types not previously listed:



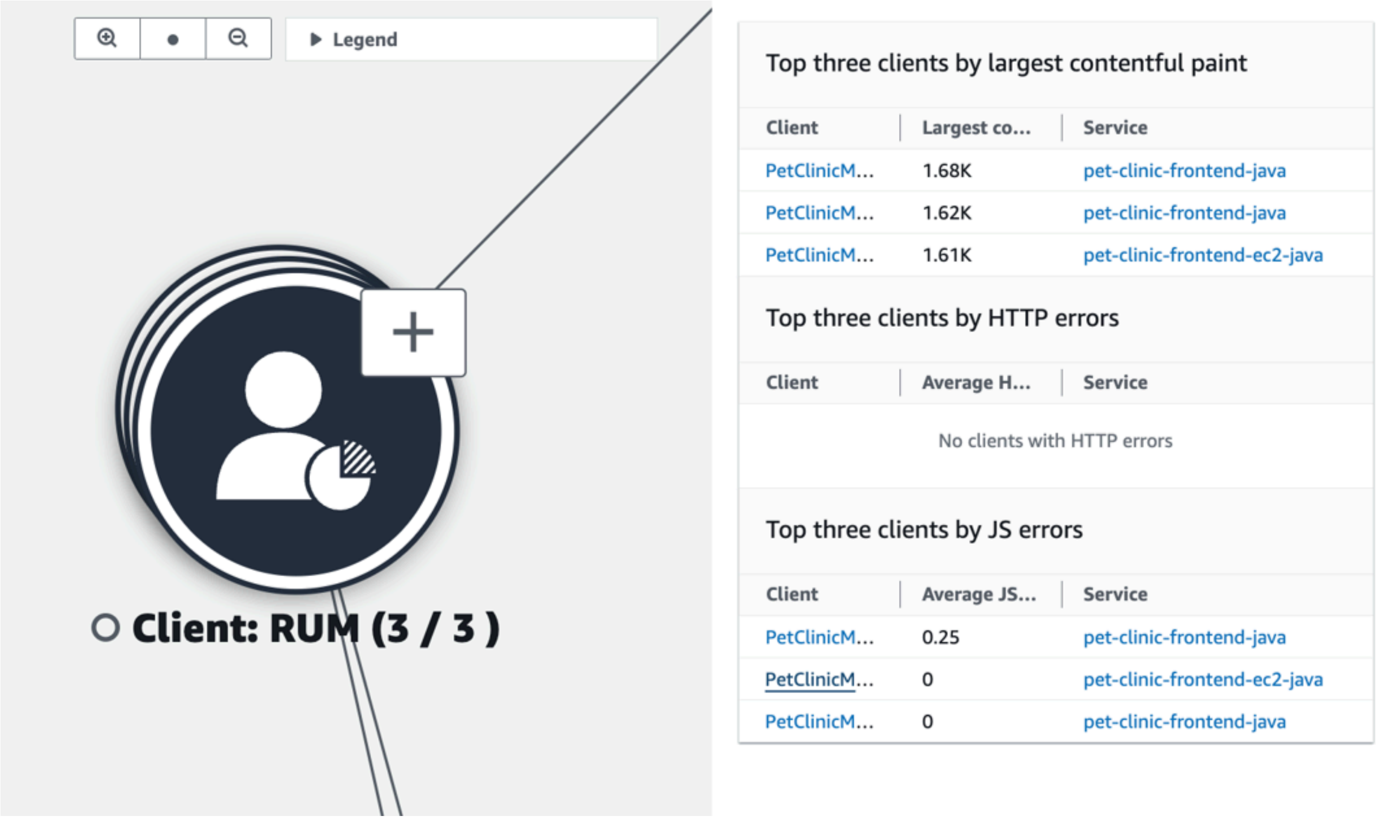
View clients

After you turn on X-Ray tracing for your CloudWatch RUM web clients, they display on the service map connected to services they call.

Choose a client node to open a pane displaying detailed client information:

- Metrics for page loads, average load time, errors, and average web vitals.
- A graph displaying a breakdown of errors.
- A link to display the client details in CloudWatch RUM.

RUM clients are grouped together by default into a single expandable icon. Select the (+) icon, as shown in the following image, to expand the group and see its individual elements.



The following icon represents an example of a RUM client in the service map:

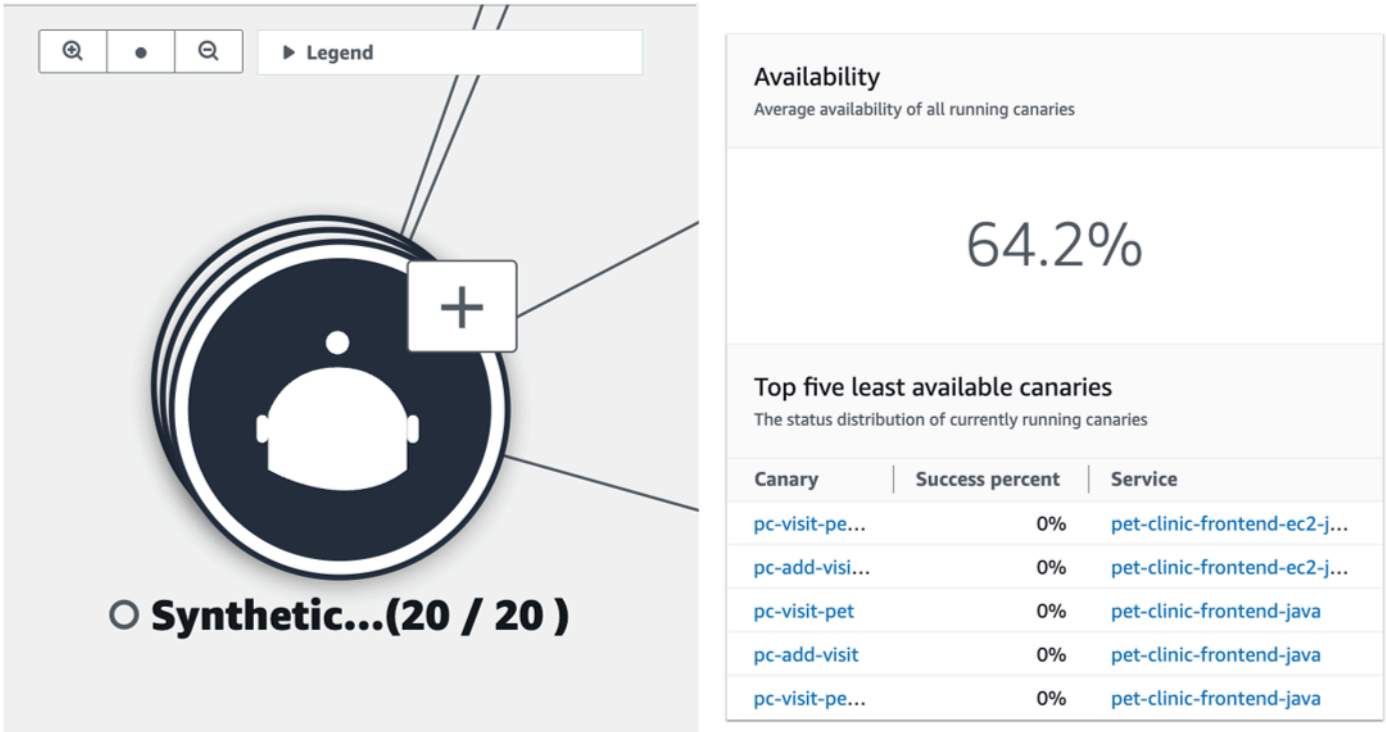
- A RUM client –

> **Note**
>
> To see AJAX errors within your client pages, use the CloudWatch RUM web client version 1.15 or newer.

View synthetics canaries

After you turn on Amazon X-Ray tracing for your CloudWatch Synthetics canaries, they display on the service map connected to services they call, as shown in the following example image:

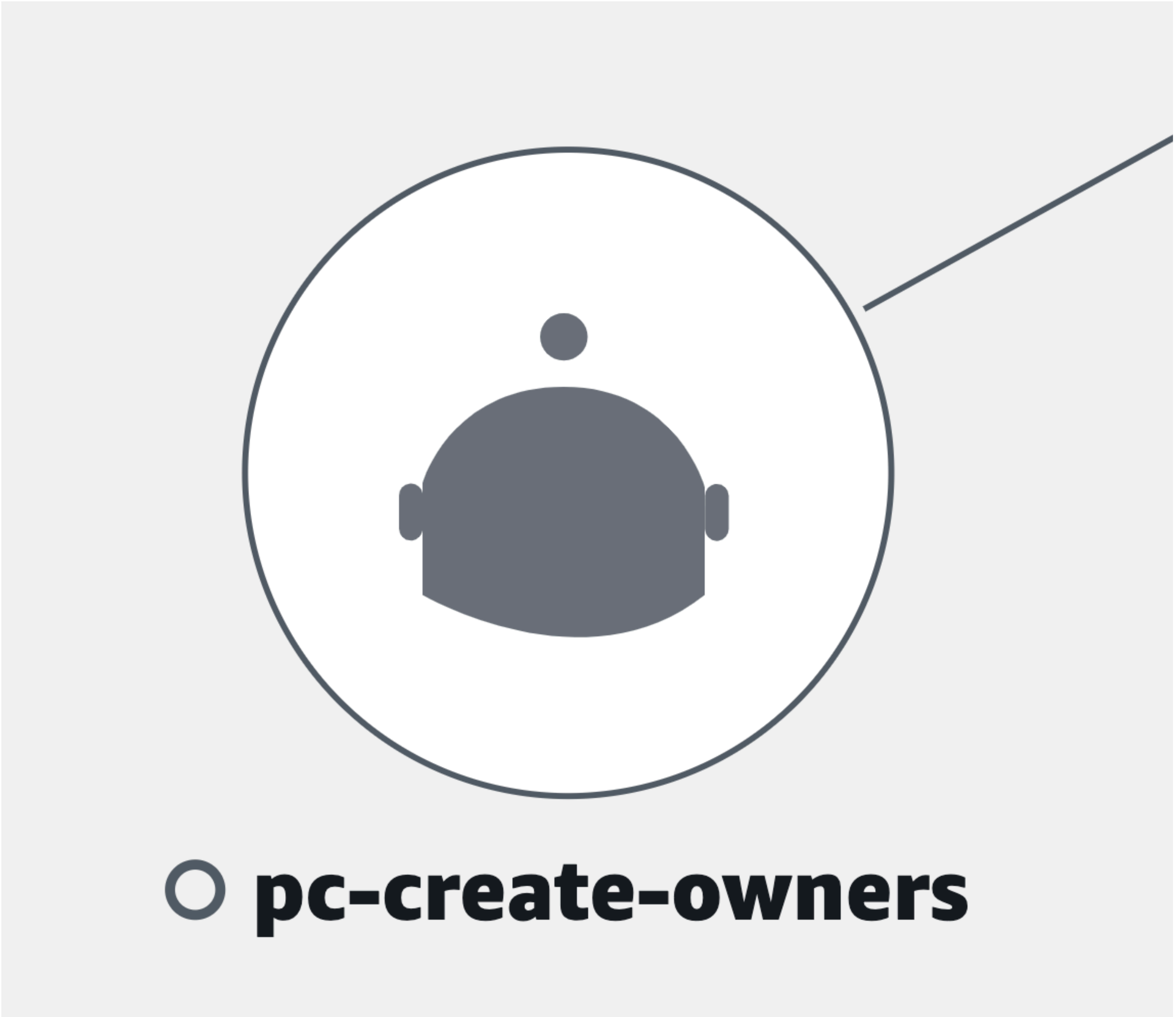
Choose a canary node to open a pane displaying detailed canary information, as shown in the following image:

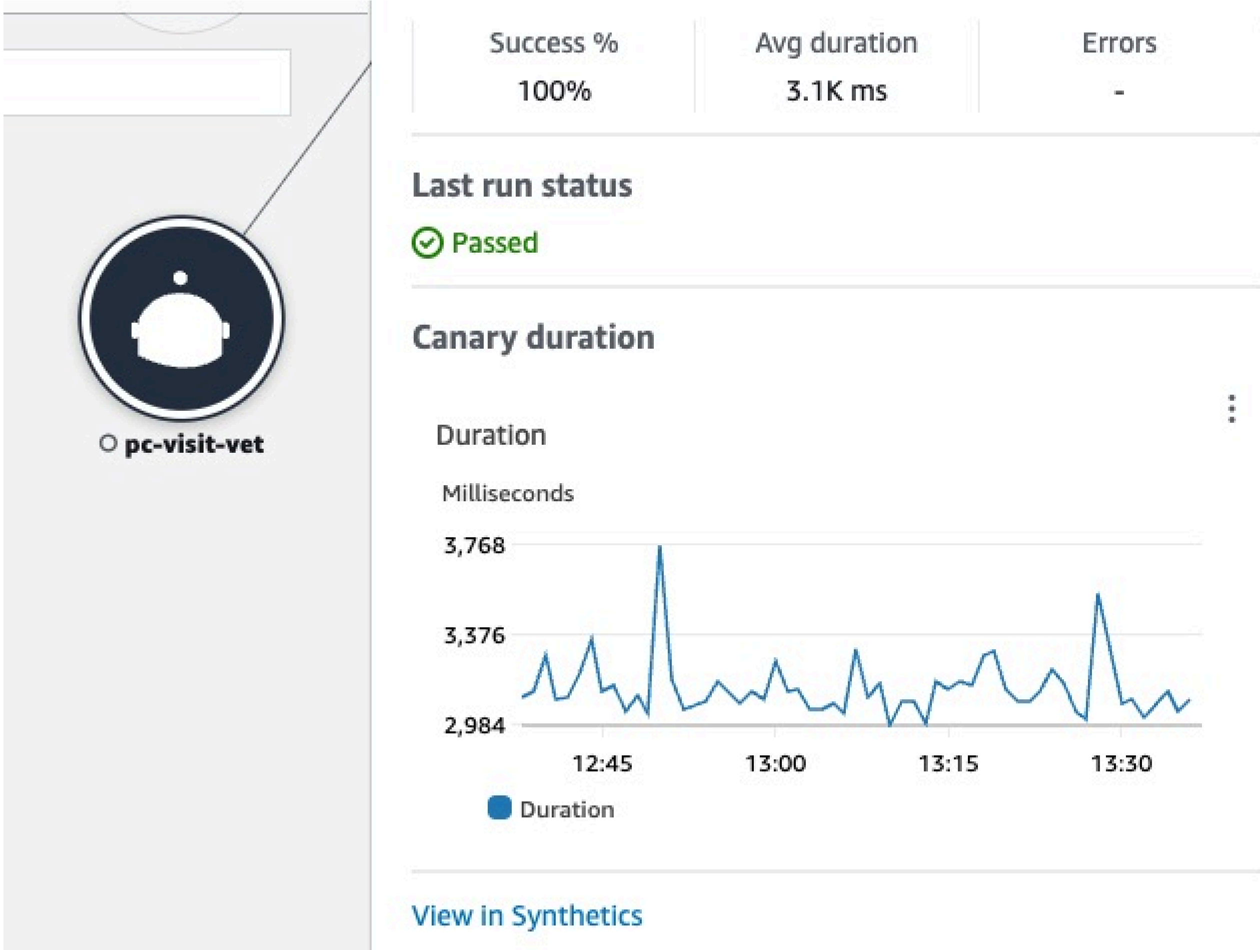


Canaries are grouped together by default into a single expandable icon. Select the (+) icon, as shown in the previous image, to expand the group and see its individual elements.

The following icons represent examples of clients in the service map:

- A synthetics canary –


pc-create-owners



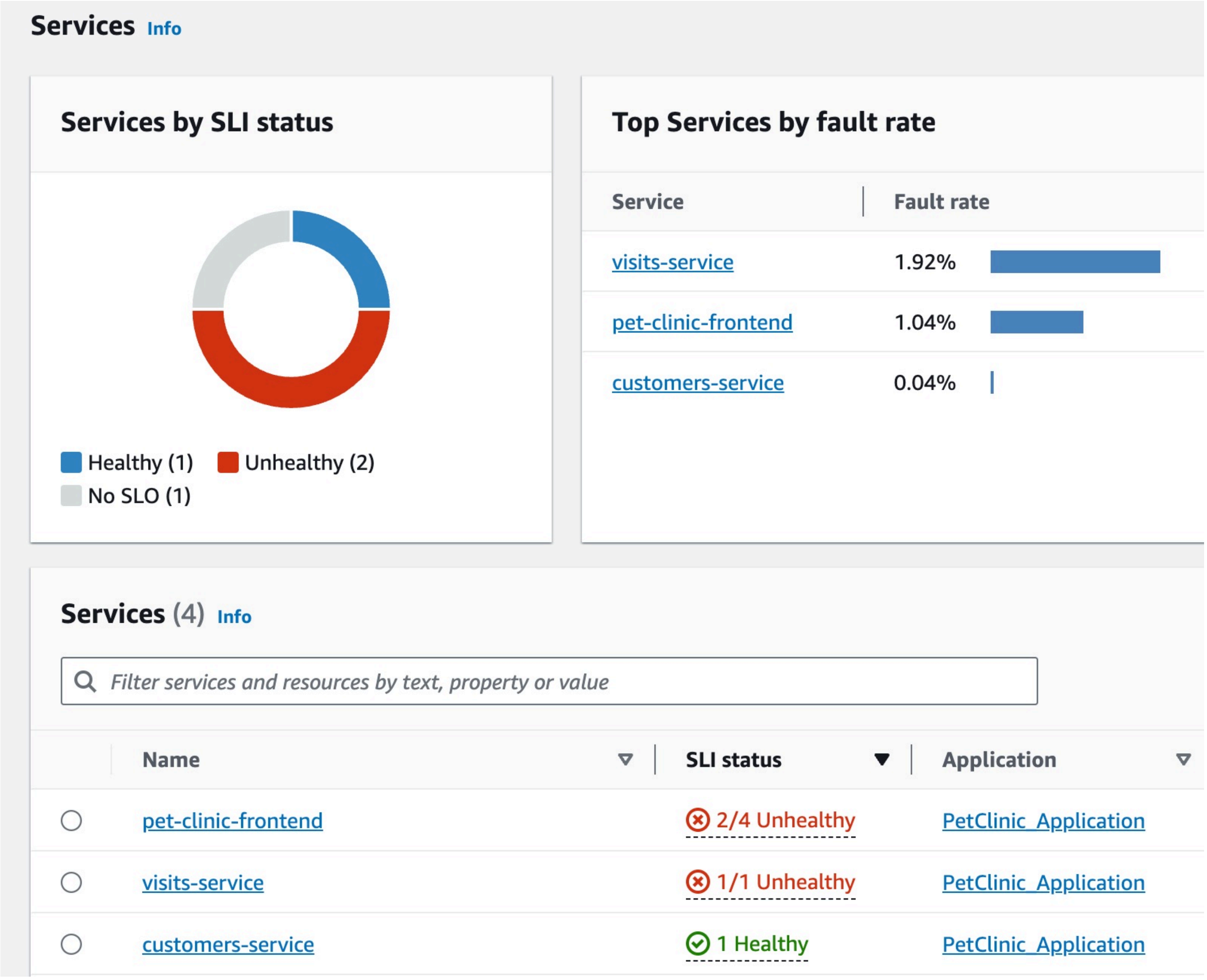
In the pane for canary nodes, you can see the following:

- Metrics for success percentage, average duration, and errors.
- The status of the last canary run.
- A graph displaying canary run duration. Hover over a graph series to see a pop-up containing more information.
- A link to display canary details in **CloudWatch Synthetics**.

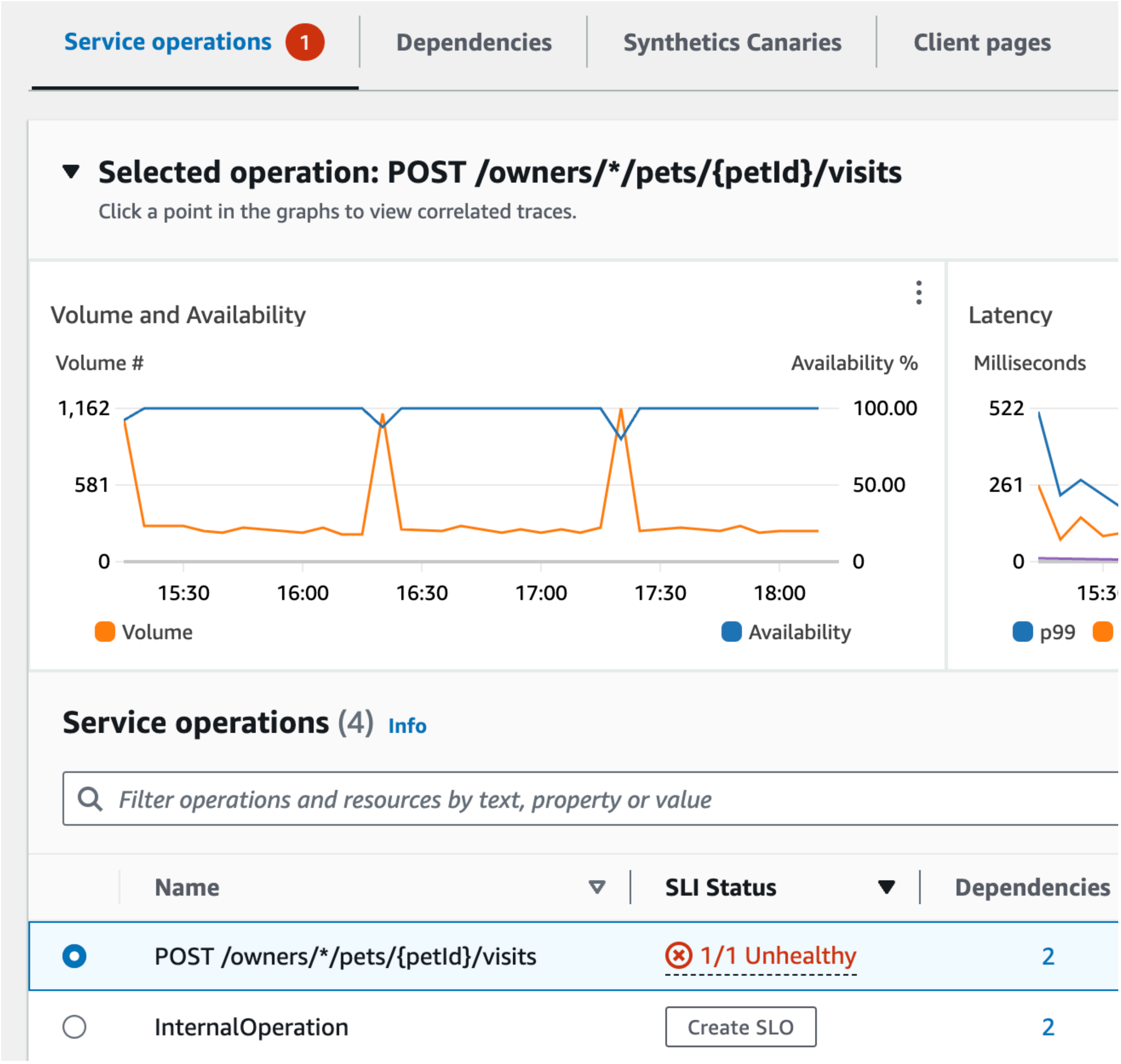
Example: Use Application Signals to resolve an operational health issue

The following scenario provides an example of how Application Signals can be used to monitor your services and identify service quality issues. Drill down to identify potential root causes and take action to resolve the issue. This example is focused on a pet clinic application composed of several microservices that call Amazon Web Services services such as DynamoDB.

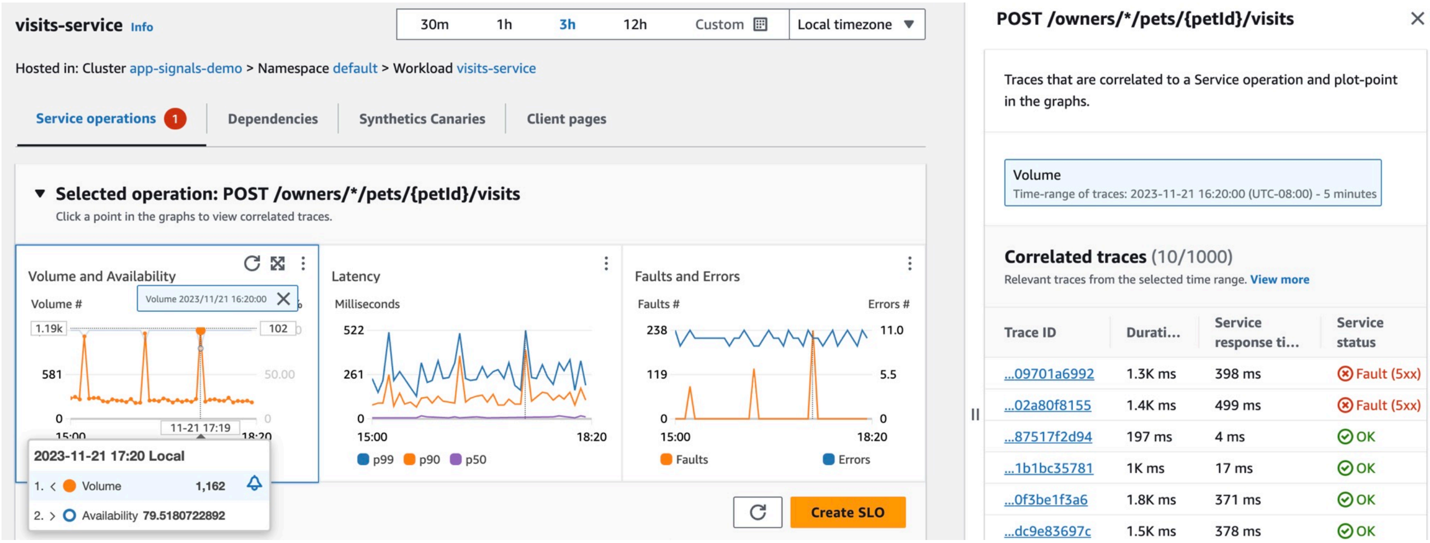
Jane is part of a DevOps team that oversees the operational health of a pet clinic application. Jane's team is committed to ensuring that the application is highly available and responsive. They use service level objectives (SLOs) to measure application performance against these business commitments. She receives an alert about several unhealthy service level indicators (SLIs). She opens the CloudWatch console and navigates to the Services page, where she sees several services in an unhealthy state.



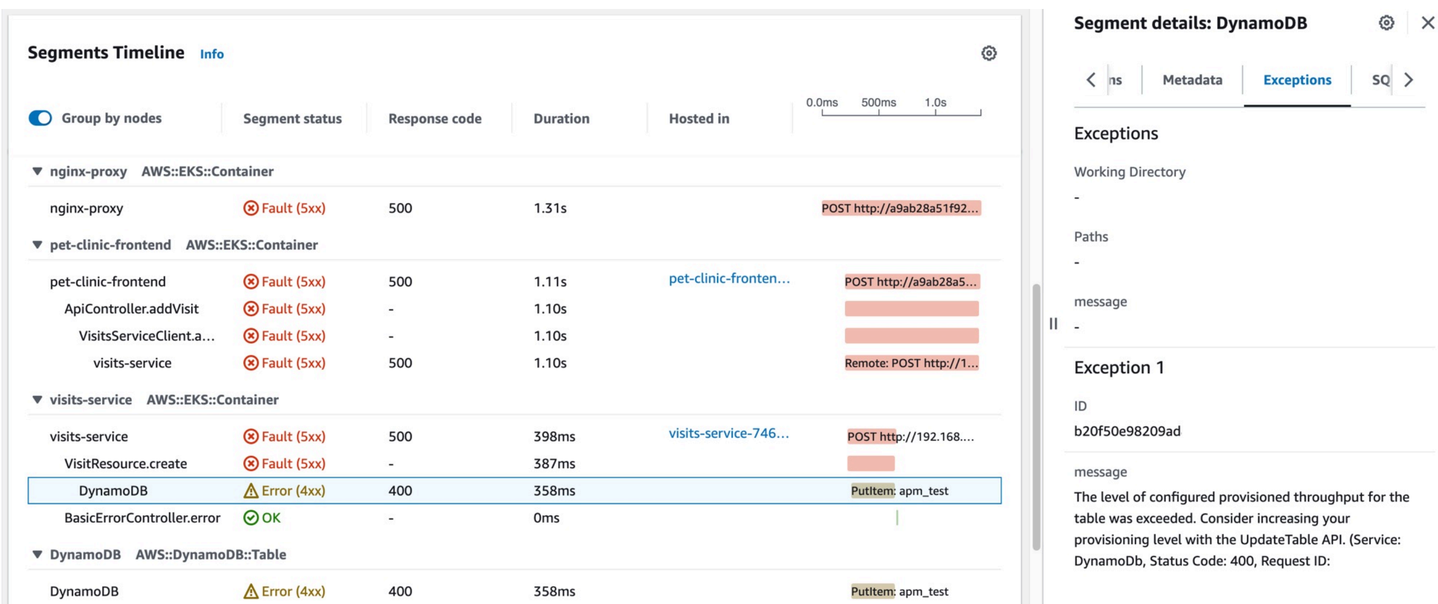
At the top of the page, Jane sees that the `visits-service` is the top service by fault rate. She selects the link in the graph, which opens the Service detail page for the service. She sees that there is an unhealthy operation in the Service operations table. She selects this operation and sees in the Volume and Availability graph that there are periodic call volume spikes that seem to correlate to dips in availability.



In order to look closer at the dips in service availability, Jane selects one of the availability data points in the graph. A drawer opens showing X-Ray traces that are correlated to the selected data point. She sees that there are multiple traces containing faults.



Jane selects one of the correlated traces with a fault status, which opens the X-Ray Trace detail page for the selected trace. Jane scrolls down to the Segments Timeline section and follows the call path until she sees that calls to a DynamoDB table are returning errors. She selects the DynamoDB segment and navigates to the Exceptions tab of the right-side drawer.



Jane sees that a DynamoDB resource is misconfigured, resulting in errors during spikes in customer requests. The DynamoDB table's level of provisioned throughput is periodically exceeded, resulting in service availability issues and unhealthy SLIs. Based on this information, her team is able to configure a higher level of provisioned throughput and ensure high availability of the application.

Example: Use Application Signals to troubleshoot generative AI applications interacting with Amazon Bedrock models

You can use Application Signals to troubleshoot your generative AI applications that interact with Amazon Bedrock models. Application Signals streamlines this process by providing out-of-the-box telemetry data, offering deeper insights into your application's interactions with LLM models. It helps address key use cases such as:

- Model configuration issues
- Model usage costs
- Model latency
- Model response generation stopped reasons

Enabling Application Signals with LLM/GenAI Observability provides real-time visibility into your application's interactions with Amazon Bedrock services. Application Signals automatically generates and correlates performance metrics and traces for Amazon Bedrock API calls.

Application Signals currently support the following LLM Models from Amazon Bedrock.

- AI21 Jamba
- Amazon Titan
- Anthropic Claude
- Cohere Command
- Meta Llama
- Mistral AI
- Nova

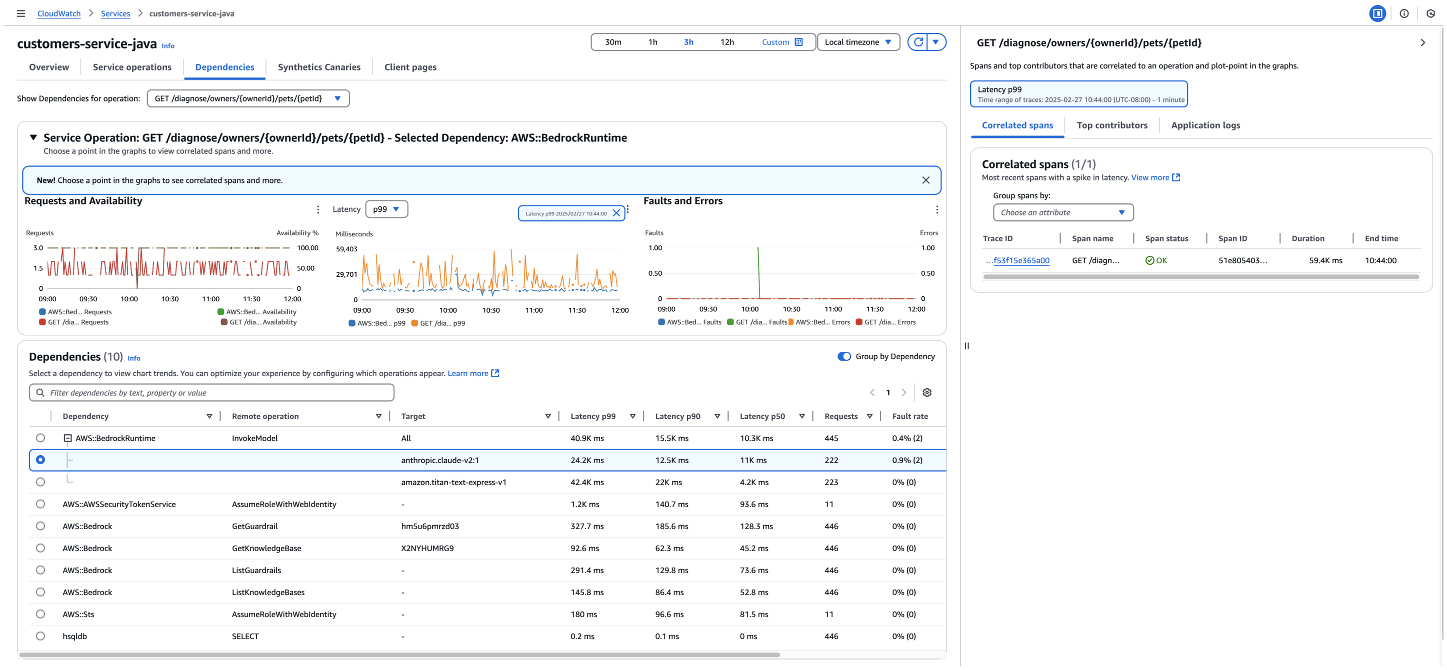
Fine-grained metrics and traces

For each Amazon Bedrock API call, Application Signals generates detailed performance metrics at the resource level, including:

- Model ID
- Guardrails ID
- Knowledge Base ID

- Bedrock Agent ID

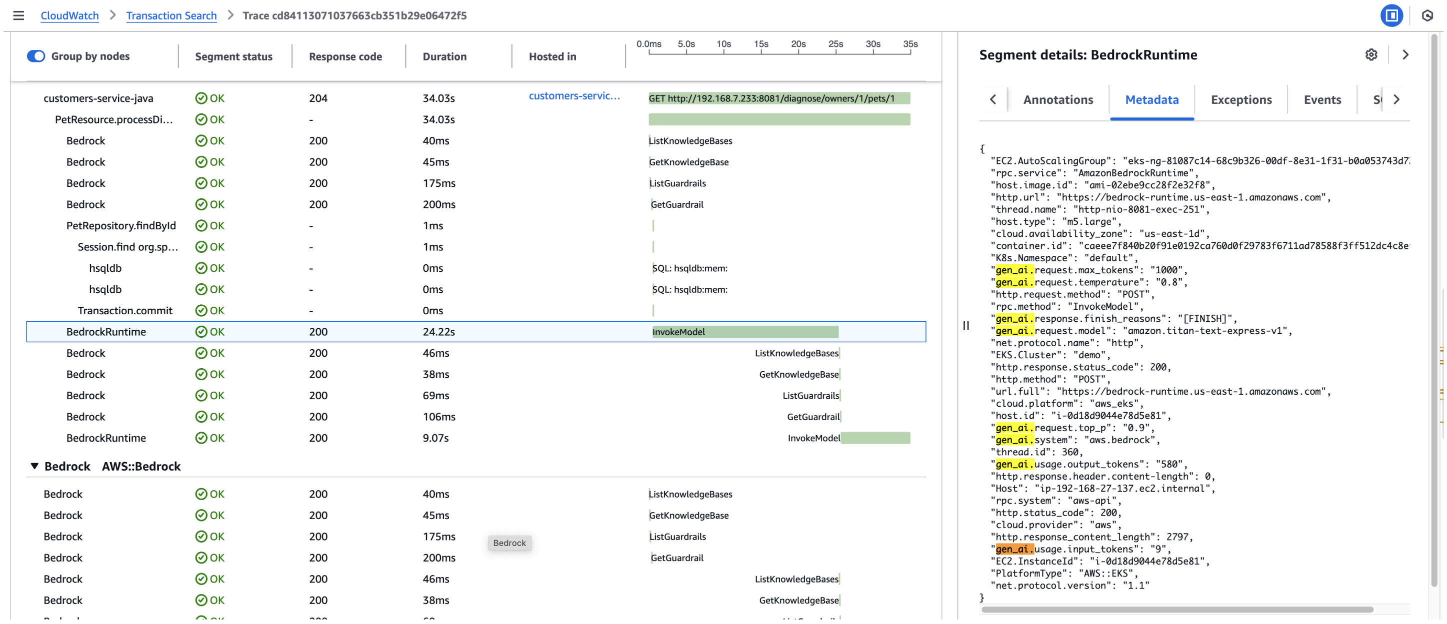
Additionally, correlated trace spans at the same level help provide a comprehensive view of request execution and dependencies.



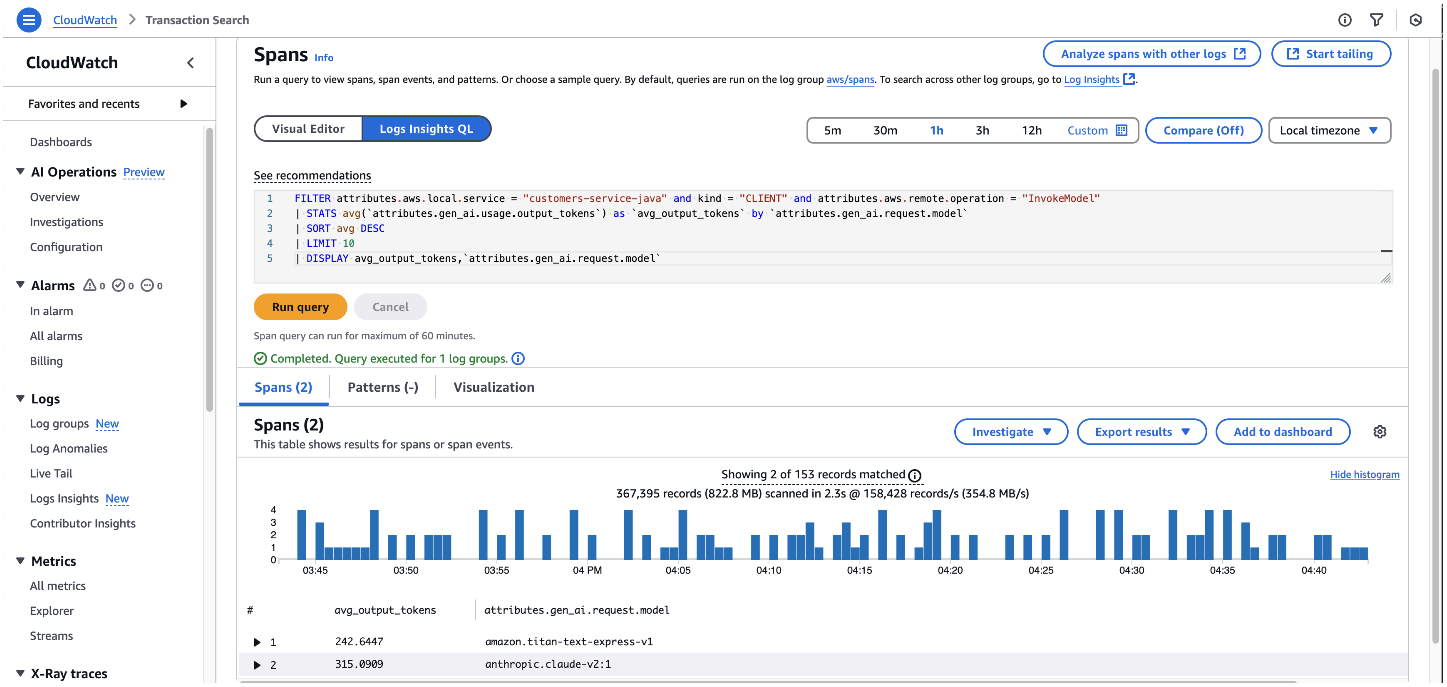
OpenTelemetry GenAI attributes support

Application Signals generates the following GenAI attributes for Amazon Bedrock API calls with OpenTelemetry semantic convention. These attributes help analyze model usage, cost, and response quality, and can be leveraged through Transaction Search for deeper insights.

- gen_ai.system
- gen_ai.request.model
- gen_ai.request.max_tokens
- gen_ai.request.temperature
- gen_ai.request.top_p
- gen_ai.usage.input_tokens
- gen_ai.usage.output_tokens
- gen_ai.response.finish_reasons



For example, your can leverage the analytic capability from Transaction Search to compare the token usage and cost across different LLM models for the same prompt, enabling cost-efficient model selection.



For more information, see [Improve Amazon Bedrock Observability with CloudWatch Application Signals](#).

Metrics collected by Application Signals

Application Signals collects both standard application metrics and runtime metrics from the applications that you enable it for.

Standard application metrics relate to the most critical parts of service performance, latency and availability.

Runtime metrics track application metrics over time, including memory usage, CPU usage, and garbage collection. Application Signals displays the runtime metrics in the context of the services that you have enabled for Application Signals. When you have an operational issue, observing the runtime metrics can be useful to help you find the root cause of the issue. For example, you can see if spikes in latency in your service relate to spikes in a runtime metric.

Topics

Standard application metrics collected

Application Signals collects *standard application metrics* from the services that it discovers. These metrics relate to the most critical aspects of a service's performance: latency, faults, and errors. They can help you identify issues, monitor performance trends, and optimize resources to improve the overall user experience.

The following table lists the metrics collected by Application Signals. These metrics are sent to CloudWatch in the `ApplicationSignals` namespace.

Metric	Description
`Latency`	The delay before data transfer begins after the request is made. Units: Milliseconds
`Fault`	A count of both HTTP 5XX server-side faults and OpenTelemetry span status errors.

Metric	Description
	Units: None
`Error`	A count of HTTP 4XX client-side errors. These are considered to be request errors that are not caused by service problems. Therefore, the `Availability` metric displayed on Application Signals dashboards does not regard these errors as service faults. Units: None

The `Availability` metric displayed on Application Signals dashboards is computed as **(1 - `Faults/Total`)*100**. Total responses includes all responses and is derived from `SampleCount(Latency)`. Successful responses are all responses without a `5XX` error. `4XX` responses are treated as successful when Application Signals calculates `Availability`.

Dimensions collected and dimension combinations

The following dimensions are defined for each of the standard application metrics. For more information about dimensions, see Dimensions.

Different dimensions are collected for *service metrics* and *dependency metrics*. Within the services discovered by Application Signals, when microservice A calls microservice B, microservice B is serving the request. In this case, microservice A emits dependency metrics and microservice B emits service metrics. When a client calls microservice A, microservice A is serving the request and emits service metrics.

Dimensions for service metrics

The following dimensions are collected for service metrics.

Dimension	Description
`Service`	The name of the service. The maximum value is 255 characters.
`Operation`	The name of the API operation or other activity.

Dimension	Description
	The maxiumum value is 1024 characters. Currently, you can set service level objectives on operations only if the operation name is 194 characters or fewer.
`Environment`	The name of the environment where services are running. If services are not running on Amazon EKS, you can specify an optional custom value for `deployment.environment` in the `OTEL_ATTRIBUTE_RESOURCES` parameter. The maxiumum value is 259 characters.

When you view these metrics in the CloudWatch console, you can view them using the following dimension combinations:

- `[Environment, Service, Operation, [Latency, Error, Fault]]`
- `[Environment, Service, [Latency, Error, Fault]]`

Dimensions for dependency metrics

The following dimensions are collected for dependency metrics:

Dimension	Description
`Service`	The name of the service. The maximum value is 255 characters.
`Operation`	The name of the API operation or other operation. The maximum value is 1024 characters.
`RemoteService`	The name of the remote service being invoked. The maximum value is 255 characters.
`RemoteOperation`	The name of the API operation being invoked.

Dimension	Description
	The maximum value is 1024 characters.
`Environment`	The name of the environment where services are running. If services are not running on Amazon EKS, you can specify an optional custom value for `deployment.environment` in the `OTEL_ATTRIBUTE_RESOURCES` parameter. The maxiumum value is 259 characters.
`RemoteEnvironment`	The name of the environment where dependency services are running. The `RemoteEnvironment` parameter is automatically generated when a service calls a dependency and they are both running in the same cluster. Otherwise, `RemoteEnvironment` is neither generated nor reported in the service dependency's metrics. Currently only available on Amazon EKS and K8S platforms. The maximum value is 259 characters.
`RemoteResourceIdentifier`	The name of the resource invoked by a remote call. The `RemoteResourceIdentifier` parameter is automatically generated if service calls a remote Amazon service. Otherwise, `RemoteResourceIdentifier` is neither generated nor reported in the service dependency's metrics. The maximum value is 1024 characters.
`RemoteResourceType`	The type of the resource that is invoked by a remote call. Required only if `RemoteResourceIdentifier` is defined. The maximum value is 1024 characters.

When you view these metrics in the CloudWatch console, you can view them using the following dimension combinations:

Running on Amazon EKS clusters

- `[Environment, Service, Operation, RemoteService, RemoteOperation, RemoteEnvironment, RemoteResourceIdentifier, RemoteResourceType, [Latency, Error, Fault]]`
- `[Environment, Service, Operation, RemoteService, RemoteOperation, RemoteEnvironment, [Latency, Error, Fault]]`
- `[Environment, Service, Operation, RemoteService, RemoteOperation, RemoteResourceIdentifier, RemoteResourceType, [Latency, Error, Fault]]`
- `[Environment, Service, Operation, RemoteService, RemoteOperation, [Latency, Error, Fault]]`
- `[Environment, Service, RemoteService, RemoteEnvironment, [Latency, Error, Fault]]`
- `[Environment, Service, RemoteService, [Latency, Error, Fault]]`
- `[Environment, Service, RemoteService, RemoteOperation, RemoteEnvironment, RemoteResourceIdentifier, RemoteResourceType, [Latency, Error, Fault]]`
- `[Environment, Service, RemoteService, RemoteOperation, RemoteEnvironment, [Latency, Error, Fault]]`
- `[Environment, Service, RemoteService, RemoteOperation, RemoteResourceIdentifier, RemoteResourceType, [Latency, Error, Fault]]`
- `[Environment, Service, RemoteService, RemoteOperation, [Latency, Error, Fault]]`
- `[RemoteService [Latency, Error, Fault]]`
- `[RemoteService, RemoteResourceIdentifier, RemoteResourceType [Latency, Error, Fault]]`

Runtime metrics

Application Signals uses the Amazon Distro for OpenTelemetry SDK to automatically collect OpenTelemetry-compatible metrics from your Java and Python applications. To have runtime metrics collected, you must meet the following pre-requisites:

- Your CloudWatch agent must be version `1.300049.1` or later.
- If you use the Amazon CloudWatch Observability EKS add-on, it must be version `2.30-eksbuild.1` or later. If you update the add-on, you must restart your applications.

- For Java applications, you must be running `1.32.5` or later of the Amazon Distro for OpenTelemetry SDK for Java.
- For Python applications, you must be running `0.7.0` or later of the Amazon Distro for OpenTelemetry SDK for Python.
- For .Net applications, you must be running `1.6.0` or later of the Amazon Distro for OpenTelemetry SDK for .Net.

Runtime metrics are not collected for Node.js applications.

Runtime metric are charged as part of Application Signals costs. For more information about CloudWatch pricing, see [Amazon CloudWatch Pricing](#).

> **Note**
>
> **Known issues**
> The runtime metrics collection in the Java SDK release v1.32.5 is known to not work with applications using JBoss Wildfly. This issue extends to the Amazon CloudWatch Observability EKS add-on, affecting versions `2.3.0-eksbuild.1` through `2.6.0-eksbuild.1`. The issue is fixed in Java SDK release `v1.32.6` and the Amazon CloudWatch Observability EKS add-on version `v3.0.0-eksbuild.1`.
> If you are impacted, either upgrade the Java SDK version or disable your runtime metrics collection by adding the environment variable `OTEL_AWS_APPLICATION_SIGNALS_RUNTIME_ENABLED=false` to your application.

Java runtime metrics

Application Signals collects the following JVM metrics from Java applications that you enable for Application Signals. All runtime metrics are sent to CloudWatch in the `ApplicationSignals` namespace, and are collected with the `Service` and `Environment` dimension set.

Metric name	Description	Meaningful statistics
`JVMGCDuration`	Aggregated metric for the duration of JVM garbage collection actions.	Sum, Average, Minimum, Maximum

Metric name	Description	Meaningful statistics
	Unit: Milliseconds	
`JVMGCOldGenDuration`	Aggregated metric for the duration of JVM garbage collection actions of the old generation. Available only in G1. Unit: Milliseconds	Sum, Average, Minimum, Maximum
`JVMGCYoungGenDuration`	Aggregated metric for the duration of JVM garbage collection actions of the young generation. Available only in G1. Unit: Milliseconds	Sum, Average, Minimum, Maximum
`JVMGCCount`	Aggregated metric for the number of JVM garbage collection actions. Unit: None	Sum, Average, Minimum, Maximum
`JVMGCOldGenCount`	Aggregated metric for the number of JVM garbage collection actions of the old generation. Available only in G1. Unit: None	Sum, Average, Minimum, Maximum
`JVMGCYoungGenCount`	Aggregated metric for the number of JVM garbage collection actions of the young generation. Available only in G1. Unit: None	Sum, Average, Minimum, Maximum
`JVMMemoryHeapUsed`	The amount of memory heap used. Unit: Bytes	Average, Minimum, Maximum

Metric name	Description	Meaningful statistics
JVMMemoryUsedAfter LastGC	Amount of memory used, as measured after the most recent garbage collection event on this pool. Unit: Bytes	Average, Minimum, Maximum
JVMMemoryOldGenUsed	The amount of memory used by the old generation. Unit: Bytes	Average, Minimum, Maximum
JVMMemorySurvivorS paceUsed	The amount of memory heap used by the survivor space. Unit: Bytes	Average, Minimum, Maximum
JVMMemoryEdenSpaceUsed	The amount of memory used by the eden space. Unit: Bytes	Average, Minimum, Maximum
JVMMemoryNonHeapUsed	The amount of non-heap memory used. Unit: Bytes	Average, Minimum, Maximum
JVMThreadCount	The number of executing threads, including both daemon and non-daemon threads. Unit: None	Sum, Average, Minimum, Maximum
JVMClassLoaded	The number of classes loaded. Unit: None	Sum, Average, Minimum, Maximum

Metric name	Description	Meaningful statistics
JVMCpuTime	The CPU time used by the process, as reported by the JVM. Unit: None (Nanoseconds)	Sum, Average, Minimum, Maximum
JVMCpuRecentUtilization	The recent CPU utilized by the process, as reported by the JVM. Unit: None	Average, Minimum, Maximum

Python runtime metrics

Application Signals collects the following metrics from Python applications that you enable for Application Signals. All runtime metrics are sent to CloudWatch in the `ApplicationSignals` namespace, and are collected with the `Service` and `Environment` dimension set.

Metric name	Description	Meaningful statistics
PythonProcessGCCount	The total number of objects currently being tracked. Unit: None	Sum, Average, Minimum, Maximum
PythonProcessGCGen0Count	The number of objects currently being tracked in Generation 0. Unit: None	Sum, Average, Minimum, Maximum
PythonProcessGCGen1Count	The number of objects currently being tracked in Generation 1. Unit: None	Sum, Average, Minimum, Maximum

Metric name	Description	Meaningful statistics
PythonProcessGCGen2Count	The number of objects currently being tracked in Generation 2. Unit: None	Sum, Average, Minimum, Maximum
PythonProcessVMSMemoryUsed	The total amount of virtual memory used by the process. Unit: Bytes	Average, Minimum, Maximum
PythonProcessRSSMemoryUsed	The total amount of non-swapped physical memory used by the process. Unit: Bytes	Average, Minimum, Maximum
PythonProcessThreadCount	The number of threads currently used by the process. Unit: None	Sum, Average, Minimum, Maximum
PythonProcessCpuTime	The CPU time used by the process. Unit: Seconds	Sum, Average, Minimum, Maximum
PythonProcessCpuUtilization	The CPU utilization of the process. Unit: None	Average, Minimum, Maximum

.Net runtime metrics

Application Signals collects the following metrics from .Net applications that you enable for Application Signals. All runtime metrics are sent to CloudWatch in the `ApplicationSignals` namespace, and are collected with the `Service` and `Environment` dimension set.

Metric name	Description	Meaningful statistics
`DotNetGCGen0Count`	The total number of garbage collection metrics tracked in Generation 0 since the process started. Unit: None	Sum, Average, Minimum, Maximum
`DotNetGCGen1Count`	The total number of garbage collection metrics tracked in Generation 1 since the process started. Unit: None	Sum, Average, Minimum, Maximum
`DotNetGCGen2Count`	The total number of garbage collection metrics tracked in Generation 2 since the process started. Unit: None	Sum, Average, Minimum, Maximum
`DotNetGCDuration`	The total amount of time paused in garbage collection since the process started. Unit: None	Sum, Average, Minimum, Maximum
`DotNetGCGen0HeapSize`	The heap size (including fragmentation) of Generation 0 observed during the latest garbage collection. **Note** This metric is only available after the first Garbage Collection is complete. Unit: Bytes	Average, Minimum, Maximum

Metric name	Description	Meaningful statistics
`DotNetGCGen1HeapSize`	The heap size (including fragmentation) of Generation 1 observed during the latest garbage collection. **Note** This metric is only available after the first Garbage Collection is complete. Unit: Bytes	Average, Minimum, Maximum
`DotNetGCGen2HeapSize`	The heap size (including fragmentation) of Generation 2 observed during the latest garbage collection. **Note** This metric is only available after the first Garbage Collection is complete. Unit: Bytes	Average, Minimum, Maximum

Metric name	Description	Meaningful statistics
`DotNetGCLOHHeapSize`	The large object heap size (including fragmentation) observed during the latest garbage collection. **Note** This metric is only available after the first Garbage Collection is complete. Unit: Bytes	Average, Minimum, Maximum
`DotNetGCPOHHeapSize`	The pinned object heap size (including fragmentation) observed during the latest garbage collection. **Note** This metric is only available after the first Garbage Collection is complete. Unit: Bytes	Average, Minimum, Maximum
`DotNetThreadCount`	The number of thread pool threads that currently exist. Unit: None	Average, Minimum, Maximum

Metric name	Description	Meaningful statistics
`DotNetThreadQueueLength`	The number of work items that are currently queued to be processed by the thread pool. Unit: None	Average, Minimum, Maximum

Disabling the collection of runtime metrics

Runtime metrics are collected by default for Java and Python applications that are enabled for Application Signals. If you want to disable the collection of these metrics, follow the instructions in this section for your environment.

Amazon EKS

To disable runtime metrics in Amazon EKS applications at the application level, add the following environment variable to your workload specification.

```
env:
    - name: OTEL_AWS_APPLICATION_SIGNALS_RUNTIME_ENABLED
      value: "false"
```

To disable runtime metrics in Amazon EKS applications at the cluster level, apply the configuration to the advanced configuration of your Amazon CloudWatch Observability EKS add-on.

```
{
  "agent": {
    "config": {
      "traces": {
        "traces_collected": {
          "application_signals": {

          }
        }
      },
      "logs": {
        "metrics_collected": {
          "application_signals": {
```

```
          }
        }
      }
    },
    "manager": {
      "autoInstrumentationConfiguration": {
        "java": {
          "runtime_metrics": {
            "enabled": false
          }
        },
        "python": {
          "runtime_metrics": {
            "enabled": false
          }
        },
        "dotnet": {
          "runtime_metrics": {
            "enabled": false
          }
        }
      }
    }
  }
}
```

Amazon ECS

To disable runtime metrics in Amazon ECS applications, add the environment variable `OTEL_AWS_APPLICATION_SIGNALS_RUNTIME_ENABLED=false` in the new task definition revision and redeploy the application.

EC2

To disable runtime metrics in Amazon EC2 applications, add the environment variable `OTEL_AWS_APPLICATION_SIGNALS_RUNTIME_ENABLED=false` before the application starts.

Kubernetes

To disable runtime metrics in Kubernetes applications at the application level, add the following environment variable to your workload specification.

```
env:
    - name: OTEL_AWS_APPLICATION_SIGNALS_RUNTIME_ENABLED
      value: "false"
```

To disable runtime metrics in Kubernetes applications at the cluster level, use the following:

```
helm upgrade ... \
--set-string
 manager.autoInstrumentationConfiguration.java.runtime_metrics.enabled=false \
--set-string
 manager.autoInstrumentationConfiguration.python.runtime_metrics.enabled=false \
-\-set-string
 manager.autoInstrumentationConfiguration.dotnet.runtime_metrics.enabled=false
```

Service level objectives (SLOs)

You can use Application Signals to create *service level objectives* for the services for your critical business operations or dependencies. By creating SLOs on these services, you will be able to track them on the SLO dashboard, giving you an at-a-glance view of your most important operations.

In addition to creating a quick view your operators can use to see the current status of critical operations, you can use SLOs to track the longer-term performance of your services, to ensure that they are meeting your expectations. If you have service level agreements with customers, SLOs are a great tool to ensure that they are met.

Assessing your services' health with SLOs starts with setting clear, measurable objectives based on key performance metrics— *service level indicators (SLIs)*. An SLO tracks the SLI performance against the threshold and goal that you set, and reports how far or how close your application performance is to the threshold.

Application Signals helps you set SLOs on your key performance metrics. Application Signals automatically collects `Latency` and `Availability` metrics for every service and operation that it discovers, and these metrics are often ideal to use as SLIs. With the SLO creation wizard, you can use these metrics for your SLOs. You can then track the status of all of your SLOs with the Application Signals dashboards.

You can set SLOs on specific operations or dependencies that your service calls or uses. You can use any CloudWatch metric or metric expression as an SLI, in addition to using the `Latency` and `Availability` metrics.

Creating SLOs is very important for getting the most benefit from CloudWatch Application Signals. After you create SLOs, you can view their status in the Application Signals console to quickly see which of your these critical services and operations are performing well and which are unhealthy. Having SLOs to track provides the following major benefits:

- It is easier for your service operators to see the current operational health of critical services as measured against the SLI. Then they can quickly triage and identify unhealthy services and operations.
- You can track your service performance against measurable business goals over longer periods of time.

By choosing what to set SLOs on, you are prioritizing what is important to you. The Application Signals dashboards automatically present information about what you have prioritized.

When you create an SLO, you can also choose to create CloudWatch alarms at the same time to monitor the SLOs. You can set alarms that monitor for breaches of the threshold, and also for warning levels. These alarms can automatically notify you if the SLO metrics are breaching the threshold that you set, or if they are nearing a warning threshold. For example, an SLO nearing its warning threshold can let you know that your team might need to slow down churn in the application to make sure that long-term performance goals are met.

Topics

SLO concepts

An SLO includes the following components:

- A *service level indicator (SLI)*, which is a key performance metric that you specify. It represents the desired level of performance for your application. Application Signals automatically collects the key metrics `Latency` and `Availability` for the services and operations that it discovers, and these can often be ideal metrics to set SLOs for.

  You choose the threshold to use for your SLI. For example, 200ms for latency.
- A *goal* or *attainment goal*, which is the percentage of time or requests that the SLI is expected to meet the threshold over each time *interval*. The time intervals can be as short as hours or as long as a year.

  Intervals can be either calendar intervals or rolling intervals.
  - *Calendar intervals* are aligned with the calendar, such as an SLO that is tracked per month. CloudWatch automatically adjusts health, budget, and attainment numbers based on the number of days in a month. Calendar intervals are better suited for business goals that are measured on a calendar-aligned basis.
  - *Rolling intervals* are calculated on a rolling basis. Rolling intervals are better suited for tracking recent user experience of your application.
- The *period* is a shorter length of time, and many periods make up an interval. The application's performance is compared to the SLI during each period within the interval. For each period, the application is determined to have either achieved or not achieved the necessary performance.

For example, a goal of 99% with a calendar interval of one day and a period of 1 minute means that the application must meet or achieve the success threshold during 99% of the 1-minute periods during the day. If it does, then the SLO is met for that day. The next day is a new evaluation interval, and the application must meet or achieve the success threshold during 99% of the 1-minute periods during the second day to meet the SLO for that second day.

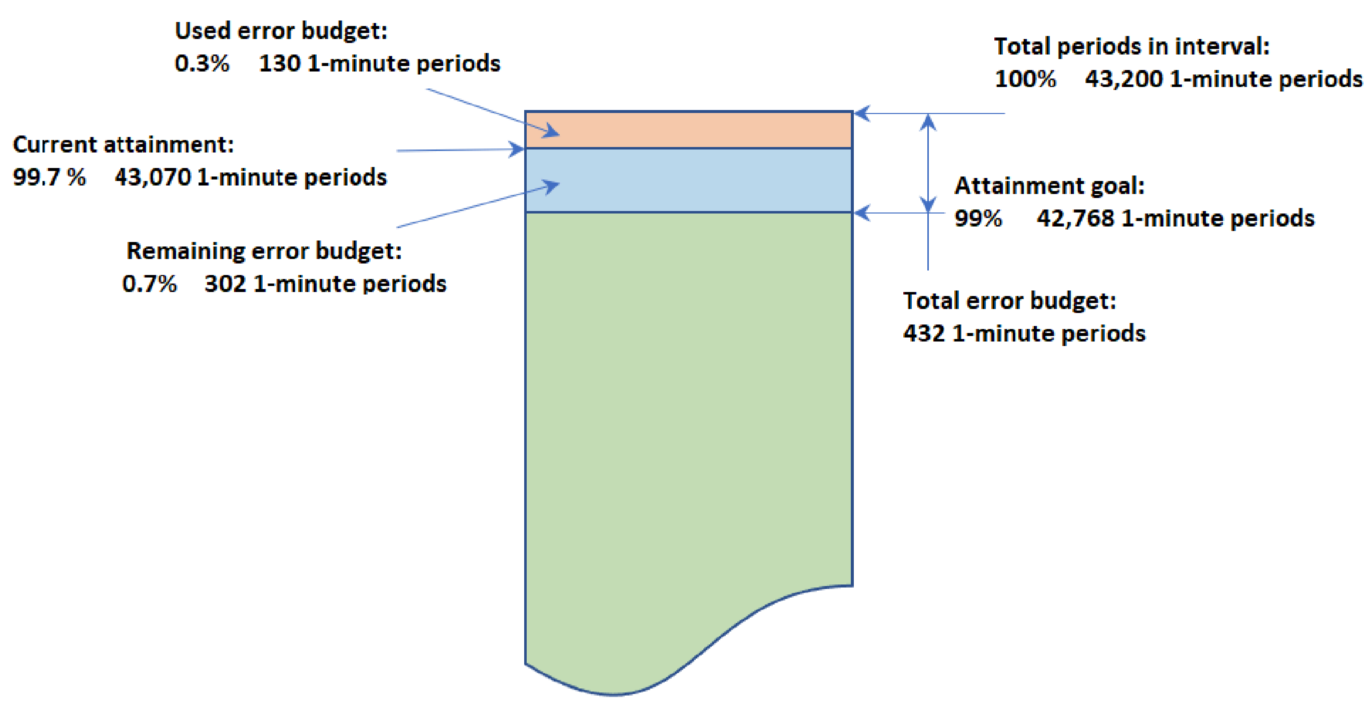
An SLI can be based on one of the new standard application metrics collected by Application Signals. Alternatively, it can be any CloudWatch metric or metric expression. The standard application metrics that you can use for an SLI are `Latency` and `Availability`. `Availability` represents the successful responses divided by the total requests. It is calculated as **(1 - Fault Rate)*100**, where Fault responses are `5xx` errors. Success responses are responses without a 5XX error. 4XX responses are treated as successful.

Calculate error budget and attainment for period-based SLOs

When you view information about an SLO, you see its current health status and its *error budget*. The error budget is the amount of time within the interval that can breach the threshold but still let the SLO be met. The *total error budget* is the total amount of breaching time that can be tolerated through the entire interval. The *remaining error budget* is the remaining amount of breaching time that can be tolerated during the current interval. This is after the amount of breaching time that has already happened has been subtracted from the total error budget.

The following figure illustrates the attainment and error budget concepts for a goal with a 30-day interval, 1-minute periods, and a 99% attainment goal. 30 days includes 43,200 1-minute periods. 99% of 43,200 is 42,768, so 42,768 minutes during the month must be healthy for the SLO to be met. So far in the current interval, 130 of the 1-minute periods were unhealthy.

SLO with an interval of 30 days and 1-minute periods



Determine success within each period

Within each period, the SLI data is aggregated into a single data point based on the statistic used for the SLI. This data point represents the entire length of the period. That single data point is compared to the SLI threshold to determine if the period is healthy. Seeing unhealthy periods during the current time range on the dashboard can alert your service operators that the service needs to be triaged.

If the period is determined to be unhealthy, the entire length of the period is counted as failed against the error budget. Tracking the error budget lets you know whether the service is achieving the performance that you want over a longer period of time.

Time window exclusions

Time window exclusions is a block of time with a defined start and end date. This time period is excluded from the SLO's performance metrics and you can schedule one-time or recurring time exclusions windows. For example, scheduled maintenance.

Note

- For period-based SLOs, SLI data in the exclusion window is considered as non-breaching.
- For request-based SLOs, all good and bad requests in the exclusion window are excluded.
- When an interval for a request-based SLO is completely excluded, a default attainment rate metric of 100% is published.
- You can only specify time windows with a start date in the future.

Calculate error budget and attainment for request-based SLOs

After you have created an SLO, you can retrieve error budget reports for it. An *error budget* is the amount of requests that your application can be non-compliant with the SLO's goal, and still have your application meet the goal. For a request-based SLO, the remaining error budget is dynamic and can increase or decrease, depending on the ratio of good requests to total requests

The following table illustrates the calculation for a request-based SLO with an interval of 5 days and 85% attainment goal. In this example, we assume there is no traffic before Day 1. The SLO did not meet the goal on Day 10.

Time	Total requests	Bad requests	Accumulative total requests in last 5 days	Accumulative total good requests in last 5 days	Request-based attainment	Total budget requests
Day 1	10	1	10	9	9/10 = 90%	1.5
Day 2	5	1	15	13	13/15=86%	2.3
Day 3	1	1	16	13	13/16=81%	2.4
Day 4	24	0	40	37	37/40=92%	6.0
Day 5	20	5	60	52	52/60=87%	9.0
Day 6	6	2	56	47	47/56=84%	8.4
Day 7	10	3	61	50	50/61=82%	9.2
Day 8	15	6	75	59	59/75=79%	11.3
Day 9	12	1	63	46	46/63=73%	9.5
Day 10		5	57	40	40/57=70%	8.5
Final attainment for last 5 days					**70%**	

Calculate burn rates and optionally set burn rate alarms

You can use Application Signals to calculate the *burn rates* for your service level objectives. A burn rate is a metric that indicates how fast the service is consuming the error budget, relative to the attainment goal of the SLO. It's expressed as a mutliple factor of the baseline error rate.

The burn rate is calculated according to the *baseline error rate*, which depends on the attainment goal. The attainment goal is the percentage of either healthy time periods or successful requests that must be achieved to meet the SLO goal. The baseline error rate is (100% - attainment goal percentage), and this number would use up the exact complete error budget at the end of the SLO's time interval. So an SLO with an attainment goal of 99% would have a baseline error rate of 1%.

Monitoring the burn rate tells us how far off we are from the baseline error rate. Again taking the example of an attainment goal of 99%, the following is true:

- **Burn rate = 1**: If the burn rate remains exactly at the baseline error rate all the time, we meet exactly the SLO goal.
- **Burn rate < 1**: If the burn rate is lower than the baseline error rate, we are on track to exceed the SLO goal.
- **Burn rate > 1**: If the burn rate is higher than baseline error rate, we have chance to fail the SLO goal.

When you create burn rates for your SLOs, you can also choose to create CloudWatch alarms at the same time to monitor the burn rates. You can set a threshold for the burn rates and the alarms can automatically notify you if the burn rate metrics are breaching the threshold that you set. For example, a burn rate nearing its threshold can let you know that the SLO is burning through the error budget faster than your team can tolerate and your team might need to slow down churn in the application to make sure that long-term performance goals are met.

Creating alarms incurs charges. For more information about CloudWatch pricing, see [Amazon CloudWatch Pricing](https://aws.amazon.com/cloudwatch/pricing/).

Calculate the burn rate

To calculate the burn rate, you must specify a *look-back window*. The look-back window is the time duration over which to measure the error rate.

```
burn rate = error rate over the look-back window / (100% - attainment goal)
```

> **Note**
>
> When there is no data for the burn rate period, Application Signals calculates the burn rate based on attainment.

The error rate is calculated as the ratio of the number of bad events over the total number of events during the burn rate window:

- For period-based SLOs, error rate is calculated as bad periods divided by total periods. The total periods represents the entirety of periods during the look-back window.
- For request-based SLOs, this is a measure of bad requests divided by total requests. The total number of requests is the number of requests during the look-back window.

The look-back window must be a multiple of the SLO period time, and must be less than the SLO interval.

Determine the appropriate threshold for a burn rate alarm

When you configure a burn rate alarm, you need to choose a value for the burn rate as the alarm threshold. The value for this threshold depends on the SLO interval length and look-back window, and depends on the which method or mental model that your team wants to adopt. There are two main methods available for determining the threshold.

Method 1: Determine the percentage of estimated total error budget your team is willing to burn in the look-back window.

If you want to get alarmed when X% of the estimated error budget is spent within the last burn rate look-back hours, the burn rate threshold is the following:

```
burn rate threshold = X% * SLO interval length / look-back window size
```

For example, 5% of a 30-day (720-hour) error budget spent over one hour requires a burn rate of `5% * 720 / 1 = 36`. Therefore, if the burn rate look-back window is1 hour, we set the burn rate threshold to be 36.

You can use the CloudWatch console to create burn rate alarms using this method. You can specify the number X, and the threshold is determined using the above formula.

The SLO interval length is determined based on the SLO interval type:

- For SLOs with a rolling interval, it’s the length of the interval in hours.
- For SLOs with a calendar-based interval:
  - If the unit is days or weeks, it’s the length of the interval in hours.

- If the unit is a month, we take 30 days as the estimated length and convert it to hours.

Method 2: Determine the time unitl budget exhaustion for the next interval

To have the alarm notify you when the current error rate in the most recent look-back window indicates that the time until budget exhaustion is less than X hours away (assuming the budget remaining is currently 100%), you can use the following formula to determine the burn rate threshold.

```
burn rate threshold = SLO interval length / X
```

We emphasize that the time until budget exhaustion (X) in the above formula assumes that the total budget remaining is currently 100%, and therefore it does not take into account the amount of budget that has already been burnt in this interval. We can also think of it as the time till budget exhaustion for the next interval.

Walkthroughs for burn rate alarms

As an example, let's take an SLO with a 28-day rolling interval. Setting a burn rate alarm for this SLO involves two steps:

1. Set the burn rate and the look-back window.
2. Crete a CloudWatch alarm that monitors the burn rate.

To get started, determine how much of the total error budget the service is willing to burn through within a specific time frame. In other words, statie your objective by using this sentence: "I want to get alerted when X% of my total error budget is consumed within M minutes."

For example, you might want to set the objective to be alerted when 2% of the total error budget is consumed within 60 minutes.

To set the burn rate, you first define the look-back window. The look-back windows is M, which in this example is 60 minutes.

Next, you create the CloudWatch alarm. When you do so, you must specify a threshold for the burn rate. If burn rate exceeds this threshold, the alarm will notify you. To find the threshold,use the following formula:

```
burn rate threshold = X% * SLO interval length/ look-back window size
```

In this example, X is 2 because we want to be alerted if 2% of the error budget is consumed within 60 minutes. The interval length is 40,320 minutes (28 days), and 60 minutes is the look-back window, so the answer is:

```
burn rate threshold = 2% * 40,320 / 60 = 13.44.
```

In this example, you would set 13.44 as the alarm threshold.

Multiple alarms with different windows

By setting up alarms on multiple look-back windows, you can quickly detect sharp error rate increases with the short window and at the same time detect smaller error rate increases that eventually deplete the error budget if they remain unnoticed.

Additionally, you could set a *composite alarm* on a burn rate with long window and on a burn rate with a short window (1/12th of the long window), and be informed only when both of the burn rates breach a threshold. This way, you can ensure that you get alerted only for situations that are still happening. For more information about composite alarms in CloudWatch, see [Combining alarms](#).

> **Note**
>
> You can set a metric alarm on a burn rate when you create the burn rate. To set a compoaite alarm on multiple burn rate alarms, you must use the instructions in [Create a composite alarm](#).

One composite alarm strategy recommended in the [Google Site Reliability Engineering workbook](#) includes three composite alarms:

- One composite alarm that watches a pair of alarms, one with a one-hour window and one with a five-minute window.
- A second composite alarm that watches a pair of alarms, one with a six-hour window and one with a 30-minute window.
- A third composite alarm that watches a pair of alarms, one with a three-day window and one with a six-hour window.

The steps to do this set up are the following:

1. Create five burn rates, with windows of five minutes, 30 minutes, one hour, six hours, and three days.
2. Create the following three pairs of CloudWatch alarms. Each pair includes one long window and one short window that is 1/12th of the long window, and the thresholds are determined by using the steps in Determine the appropriate threshold for a burn rate alarm. When you calculate the threshold for each alarm in the pair, use the longer look-back window of the pair in your calculation.
   - Alarms on the 1-hour and 5-minute burn rates (the threshold is determined by 2% of the total budget)
   - Alarms on the 6-hour and 30-minute burn rates (the threshold is determined by 5% of the total budget)
   - Alarms on the 3-day and 6-hour burn rates (the threshold is determined by 10% of the total budget)
3. For each of these pairs, create a composite alarm to get alerted when both of the individual alarms go into ALARM state. For more information about creating composite alarms, see Create a composite alarm.

   For example, if your alarms for the first pair (one-hour window and five-minute window) are named `OneHourBurnRate` and `FiveMinuteBurnRate`, the CloudWatch composite alarm rule would be `ALARM(OneHourBurnRate) AND ALARM(FiveMinuteBurnRate)`

The previous strategy is possible only for SLOs with interval length of at least three hours. For SLOs with shorter interval lengths, we recommend that you start with one pair of burn rate alarms where one alarm has a look-back window that is 1/12th of the look-back window of the other alarm. Then set a composite alarm on this pair.

Create an SLO

We recommend that you set both latency and availability SLOs on your critical applications. These metrics collected by Application Signals align with common business goals.

You can also set SLOs on any CloudWatch metric or any metric math expression that results in a single time series.

The first time that you create an SLO in your account, CloudWatch automatically creates the **AWSServiceRoleForCloudWatchApplicationSignals** service-linked role in your account, if it doesn't already exist. This service-linked role allows CloudWatch to collect CloudWatch Logs data, X-Ray trace data, CloudWatch metrics data, and tagging data from applications in your account. For more information about CloudWatch service-linked roles, see Using service-linked roles for CloudWatch.

When you create an SLO, you specify whether it is a *period-based SLO* or a *request-based SLO*. Each type of SLO has a different way of evaluating your application's performance against its attainment goal.

- A *period-based SLO* uses defined *periods* of time within a specified total time interval. For each period of time, Application Signals determines whether the application met its goal. The attainment rate is calculated as the `number of good periods/number of total periods`.

  For example, for a period-based SLO, meeting an attainment goal of 99.9% means that within your interval, your application must meet its performance goal during at least 99.9% of the time periods.
- A *request-based SLO* doesn't use pre-defined periods of time. Instead, the SLO measures `number of good requests/number of total requests` during the interval. At any time, you can find the ratio of good requests to total requests for the interval up to the time stamp that you specify, and measure that ratio against the goal set in your SLO.

Topics

Create a period-based SLO

Use the following procedure to create a period-based SLO.

To create a period-based SLO

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Service Level Objectives (SLO)**.
3. Choose **Create SLO**.
4. Enter a name for the SLO. Including the name of a service or operation, along with appropriate keywords such as latency or availability, will help you quickly identify what the SLO status indicates during triage.

5. For **Set Service Level Indicator (SLI)**, do one of the following:

   - To set the SLO on either of the standard application metrics `Latency` or `Availability`:

     a. Choose **Service Operation**.
     b. Select an account that this SLO will monitor.
     c. Select the service that this SLO will monitor.
     d. Select the operation that this SLO will monitor.
     e. For **Select a calculation method**, choose **Periods**.

        The **Select service** and **Select operation** drop-downs are populated by services and operations that have been active within the past 24 hours.
     f. Choose either **Availability** or **Latency** and then set the threshold.

   - To set the SLO on any CloudWatch metric or a CloudWatch metric math expression:

     a. Choose **CloudWatch Metric**.
     b. Choose **Select CloudWatch metric**.

        The **Select metric** screen appears. Use the **Browse** or **Query** tabs to find the metric you want, or create a metric math expression.

        After you select the metric that you want, choose the **Graphed metrics** tab and select the **Statistic** and **Period** to use for the SLO. Then choose **Select metric**.

        For more information about these screens, see Graph a metric and Add a math expression to a CloudWatch graph.
     c. For **Select a calculation method**, choose **Periods**.
     d. For **Set condition**, select a comparison operator and threshold for the SLO to use as the indicator of success.

   - To set the SLO on the dependency of a service on either of the standard application metrics `Latency` or `Availability`:

     a. Choose **Service Dependency**.
     b. Under **Select a service**, select the service that this SLO will monitor.
     c. Based on the selected service, under **Select an operation**, you can select one specific operation or select **All operations** to use the metrics from all operations of this service that calls a dependency.

   d. Under **Select a dependency**, you can search and select the required dependency for which you want to measure the reliability.

      After you select the dependency, you can view the updated graph and historical data based on the dependency.

6. If you selected **Service Operation** or **Service Dependency** in step 5, set the period length for this SLO.
7. Set the **interval** and **attainment goal** for the SLO. For more information about intervals and attainment goals and how they work together, see SLO concepts.
8. (Optional) For **Set SLO burn rates** do the following:
   - Set the length (in minutes) of the look-back window for the burn rate. For information about how to choose this length, see Walkthroughs for burn rate alarms.
   - To create more burn rates for this SLO, choose **Add more burn rates** and set the look-back window for the additional burn rates.
9. (Optional) Create burn rate alarms by doing the following:
   - Under **Set burn rate alarms** select the check box for each burn rate that you want to create an alarm for. For each of these alarms, do the following:
     - Specify the Amazon SNS topic to use for notifications when the alarm goes into ALARM state.
     - Either set a burn rate threshold or specify the percentage of the estimated total budget burnt in the last look-back window you want to stay below. If you set the percentage of estimated total budget burned, the burn rate threshold is calculated for you and used in the alarm. To either decide what threshold to set or to understand how this option is used to compute the burn rate threshold, see Determine the appropriate threshold for a burn rate alarm.
10. (Optional) Set one or more CloudWatch alarms or a warning threshold for the SLO.
    a. CloudWatch alarms can use Amazon SNS to proactively notify you if an application is unhealthy based on its SLI performance.

       To create an alarm, select one of the alarm check boxes and enter or create the Amazon SNS topic to use for notifications when the alarm goes into `ALARM` state. For more information about CloudWatch alarms, see Using Amazon CloudWatch alarms. Creating alarms incurs charges. For more information about CloudWatch pricing, see Amazon CloudWatch Pricing.

b. If you set a warning threshold, it appears in Application Signals screens to help you identify SLOs that are in danger of being unmet, even if they're currently healthy.

   To set a warning threshold, enter the threshold value in **Warning threshold**. When the SLO's error budget is lower than the warning threshold, the SLO is marked with **Warning** in several Application Signals screens. Warning thresholds also appear on error budget graphs. You can also create an **SLO warning alarm** that's based on the warning threshold.

11. (Optional) For **Set SLO time window exclusion**, do the following:

   - Under **Exclude time window**, set the time window to be excluded from the SLO performance metrics.

     You can choose **Set time window** and enter the **Start window** for every hour or month or you can choose **Set time window with CRON** and enter the CRON expression.
   - Under **Repeat**, set if this time window exclusion is recurring or not.
   - (Optional) Under **Add reason**, you can choose to enter a reason for the time window exclusion. For example, scheduled maintenance.
   - Select **Add time window** to add upto 10 time exclusion windows.

12. To add tags to this SLO, choose the **Tags** tab and then choose **Add new tag**. Tags can help you manage, identify, organize, search for, and filter resources. For more information about tagging, see Tagging your Amazon resources.

   > **Note**
   >
   > If the application this SLO is related to is registered in Amazon Service Catalog AppRegistry, you can use the `awsApplication` tag to associate this SLO with that application in AppRegistry. For more information, see What is AppRegistry?

13. Choose **Create SLO**. If you also chose to create one or more alarms, the button name changes to reflect this.

Create a request-based SLO

Use the following procedure to create a request-based SLO.

To create a request-based SLO

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Service Level Objectives (SLO)**.
3. Choose **Create SLO**.
4. Enter a name for the SLO. Including the name of a service or operation, along with appropriate keywords such as latency or availability, will help you quickly identify what the SLO status indicates during triage.
5. For **Set Service Level Indicator (SLI)**, do one of the following:
   - To set the SLO on either of the standard application metrics `Latency` or `Availability`:
     a. Choose **Service Operation**.
     b. Select the service that this SLO will monitor.
     c. Select the operation that this SLO will monitor.
     d. For **Select a calculation method**, choose **Requests**.
     e. The **Select service** and **Select operation** drop-downs are populated by services and operations that have been active within the past 24 hours.
     f. Choose either **Availability** or **Latency**. If you choose **Latency**, set the threshold.
   - To set the SLO on any CloudWatch metric or a CloudWatch metric math expression:
     a. Choose **CloudWatch Metric**.
     b. For **Define target requests**, do the following:
        i. Choose whether you want to measure **Good Requests** or **Bad Requests**.
        ii. Choose **Select CloudWatch metric**. This metric will be the numerator of the ratio of target requests to total requests. If you use a latency metric, use the **Trimmed count (TC)** statistics. If the threshold is 9 ms and you're using the less than (<) comparison operator, then use threshold TC (:threshold - 1). For more information about TC, see Syntax.

            The **Select metric** screen appears. Use the **Browse** or **Query** tabs to find the metric you want, or create a metric math expression.
     c. For **Define total requests**, choose the CloudWatch metric that you want to use for the source. This metric will be the denominator of the ratio of target requests to total requests.

The **Select metric** screen appears. Use the **Browse** or **Query** tabs to find the metric you want, or create a metric math expression.

After you select the metric that you want, choose the **Graphed metrics** tab and select the **Statistic** and **Period** to use for the SLO. Then choose **Select metric**.

If you use a latency metric which emits one data point per request, use the **Sample count statistics** to count the number of total requests.

For more information about these screens, see Graph a metric and Add a math expression to a CloudWatch graph.

- To set the SLO on the dependency of a service on either of the standard application metrics `Latency` or `Availability`:

    a. Choose **Service Dependency**.

    b. Under **Select a service**, select the service that this SLO will monitor.

    c. Based on the selected service, under **Select an operation**, you can select one specific operation or select **All operations** to use the metrics from all operations of this service that calls a dependency.

    d. Under **Select a dependency**, you can search and select the required dependency for which you want to measure the reliability.

    After you select the dependency, you can view the updated graph and historical data based on the dependency.

6. Set the **interval** and **attainment goal** for the SLO. For more information about intervals and attainment goals and how they work together, see SLO concepts.

7. (Optional) For **Set SLO burn rates** do the following:

    - Set the length (in minutes) of the look-back window for the burn rate. For information about how to choose this length, see Walkthroughs for burn rate alarms.
    - To create more burn rates for this SLO, choose **Add more burn rates** and set the look-back window for the additional burn rates.

8. (Optional) Create burn rate alarms by doing the following:

    - Under **Set burn rate alarms** select the check box for each burn rate that you want to create an alarm for. For each of these alarms, do the following:

- Specify the Amazon SNS topic to use for notifications when the alarm goes into ALARM state.
- Either set a burn rate threshold or specify the percentage of the estimated total budget burnt in the last look-back window you want to stay below. If you set the percentage of estimated total budget burned, the burn rate threshold is calculated for you and used in the alarm. To either decide what threshold to set or to understand how this option is used to compute the burn rate threshold, see Determine the appropriate threshold for a burn rate alarm.

9. (Optional) Set one or more CloudWatch alarms or a warning threshold for the SLO.

   a. CloudWatch alarms can use Amazon SNS to proactively notify you if an application is unhealthy based on its SLI performance.

      To create an alarm, select one of the alarm check boxes and enter or create the Amazon SNS topic to use for notifications when the alarm goes into `ALARM` state. For more information about CloudWatch alarms, see Using Amazon CloudWatch alarms. Creating alarms incurs charges. For more information about CloudWatch pricing, see Amazon CloudWatch Pricing.

   b. If you set a warning threshold, it appears in Application Signals screens to help you identify SLOs that are in danger of being unmet, even if they're currently healthy.

      To set a warning threshold, enter the threshold value in **Warning threshold**. When the SLO's error budget is lower than the warning threshold, the SLO is marked with **Warning** in several Application Signals screens. Warning thresholds also appear on error budget graphs. You can also create an **SLO warning alarm** that's based on the warning threshold.

10. (Optional) For **Set SLO time window exclusion**, do the following:

    - Under **Exclude time window**, set the time window to be excluded from the SLO performance metrics.

      You can choose **Set time window** and enter the **Start window** for every hour or month or you can choose **Set time window with CRON** and enter the CRON expression.
    - Under **Repeat**, set if this time window exclusion is recurring or not.
    - (Optional) Under **Add reason**, you can choose to enter a reason for the time window exclusion. For example, scheduled maintenance.
    - Select **Add time window** to add upto 10 time exclusion windows.

11. To add tags to this SLO, choose the **Tags** tab and then choose **Add new tag**. Tags can help you manage, identify, organize, search for, and filter resources. For more information about tagging, see [Tagging your Amazon resources](#).

> **Note**
>
> If the application this SLO is related to is registered in Amazon Service Catalog AppRegistry, you can use the `awsApplication` tag to associate this SLO with that application in AppRegistry. For more information, see [What is AppRegistry?](#)

12. Choose **Create SLO**. If you also chose to create one or more alarms, the button name changes to reflect this.

View and triage SLO status

You can quickly see the health of your SLOs using either the **Service Level Objectives** or the **Services** options in the CloudWatch console. The **Services** view provides an at-a-glance view of the ratio of unhealthy services, calculated based on SLOs that you have set. For more information about using the **Services** option, see [Monitor the operational health of your applications with Application Signals](#).

The **Service Level Objectives** view provides a macro view of your organization. You can see the met and unmet SLOs as a whole. This gives you a view of how many of your services and operations are performing to your expectations over longer periods of time, according to the SLIs that you chose.

To view all of your SLOs using the Service Level Objectives view

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Service Level Objectives (SLO)**.

   The **Service Level Objectives (SLO)** list appears.

   You can quickly see the current status of your SLOs in the **SLI status** column. To sort the SLOs so that all the unhealthy ones are at the top of the list, choose the **SLI status** column until the unhealthy SLOs are all at the top.

The SLO table has the following default columns. You can adjust which columns are displayed by choosing the gear icon above the list. For more information about goals, SLIs, attainment, and intervals, see SLO concepts.

- The name of the SLO.
- The **Goal** column displays the percentage of periods during each interval that must successfully meet the SLI threshold for the SLO goal to be met. It also displays the interval length for the SLO.
- The **SLI status** displays whether the current operational state of the application is healthy or not. If any period during the currently selected time range was unhealthy for the SLO, then the **SLI status** displays **Unhealthy**.
- If this SLO is configured to monitor a dependency, the **Dependency** and **Remote Operation** columns will show the details about that dependency relationship.
- The **Ending attainment** is the attainment level achieved as of the end of the selected time range. Sort by this column to see the SLOs that are most in danger of not being met.
- The **Attainment delta** is the difference in attainment level between the start and end of the selected time range. A negative delta means that the metric is trending in a downward direction. Sort by this column to see the latest trends of the SLOs.
- The **Ending error budget (%)** is the percentage of total time in the period that can have unhealthy periods and still have the SLO be achieved successfully. If you set this to 5%, and the SLI is unhealthy in 5% or fewer of the remaining periods in the interval, the SLO is still achieved successfully.
- The **Error budget delta** is the difference in error budget between the start and end of the selected time range. A negative delta means that the metric is trending in a failing direction.
- The **Ending error budget (time)** is the amount of actual time in the interval that can be unhealthy and still have the SLO be achieved successfully. For example, if this is 14 minutes, then if the SLI is unhealthy for fewer than 14 minutes during the remaining interval, the SLO will still be achieved successfully.
- The **Ending error budget (requests)** is the amount of requests in the interval that can be unhealthy and still have the SLO be achieved successfully. For request-based SLOs, this value is dynamic and can fluctuate as the cumulative total number of requests changes over time.
- The **Service**, **Operation**, and **Type** columns display information about what service and operation this SLO is set for.

3. To see the attainment and error budget graphs for an SLO, choose the radio button next to the SLO name.

   The graphs at the top of the page display the **SLO attainment** and **Error budget** status. A graph about the SLI metric associated with this SLO is also displayed.

4. To further triage an SLO that is not meeting its goal, choose the service name, operation name, or dependency name associated with that SLO. You are taken to the details page where you can triage further. For more information, see View detailed service activity and operational health with the service detail page.

5. To change the time range of the charts and tables on the page, choose a new time range near the top of the screen.

Edit an existing SLO

Follow these steps to edit an existing SLO. When you edit an SLO, you can change only the threshold, interval, attainment goal, and tags. To change other aspects such as service, operation, or metric, create a new SLO instead of editing an existing one.

Changing part of an SLO core configuration, such as period or threshold, invalidates all the previous data points and assessments about attainment and health. It effectively deletes and re-creates the SLO.

> **Note**
>
> If you edit an SLO, alarms associated with that SLO are not automatically updated. You might need to update the alarms to keep them in sync with the SLO.

To edit an existing SLO

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Service Level Objectives (SLO)**.
3. Choose the radio button next to the SLO that you want to edit, and choose **Actions**, **Edit SLO**.
4. Make your changes, then choose **Save changes**.

Delete an SLO

Follow these steps to delete an existing SLO.

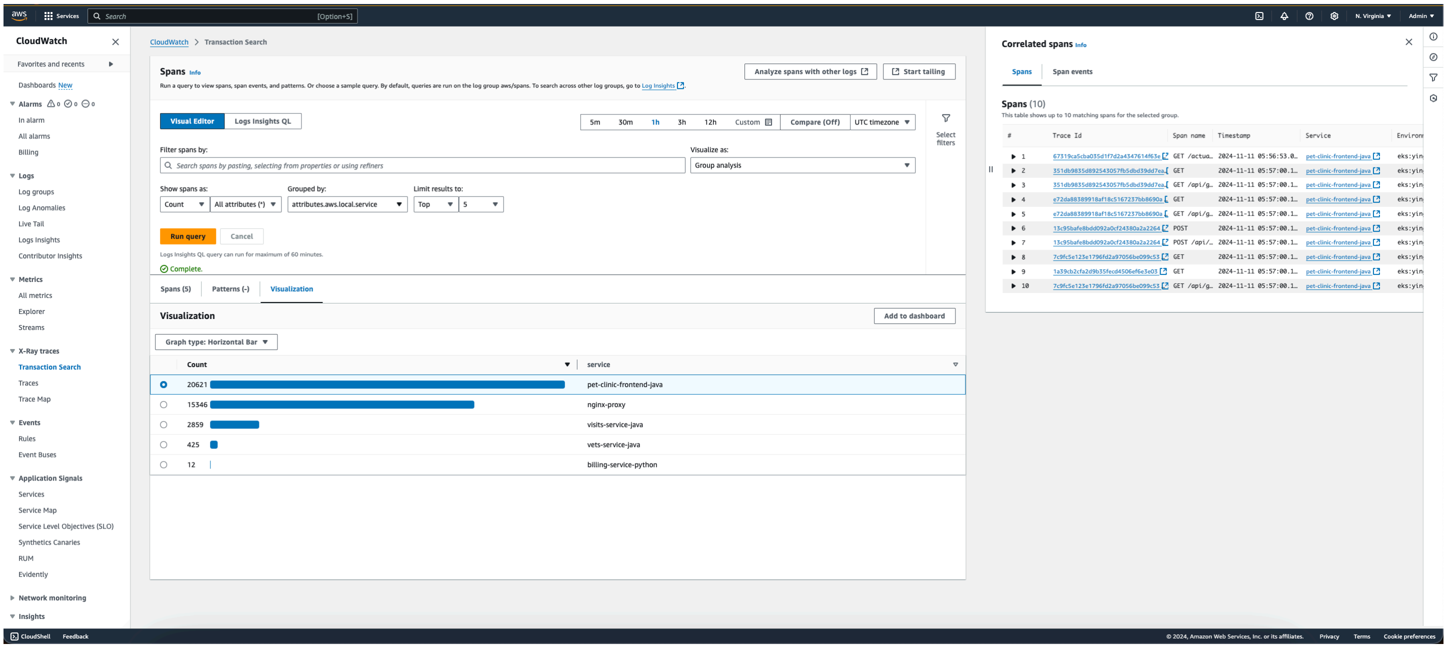
> **Note**
>
> When you delete an SLO, alarms associated with that SLO are not automatically deleted. You'll need to delete them yourself. For more information, see Managing alarms.

To delete an SLO

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Service Level Objectives (SLO)**.
3. Choose the radio button next to the SLO that you want to edit, and choose **Actions**, **Delete SLO**.
4. Choose **Confirm**.

Transaction Search

Transaction Search is an interactive analytics experience you can use to get complete visibility of your application transaction spans. Spans are the fundamental units of operation in a distributed trace and represent specific actions or tasks in an application or system. Every span records details about a particular segment of the transaction. These details include start and end times, duration, and associated metadata, which can include business attributes like customer IDs and order IDs. Spans are arranged in a parent-child hierarchy. This hierarchy forms a complete trace, mapping the flow of a transaction across different components or services.



Topics

- [Benefits]
- [How it works]
- [Pricing]
- [Getting started with Transaction Search]
- [Spans]
- [Adding custom attributes]
- [Troubleshooting application issues]

Benefits

The following are benefits of using Transaction Search:

Capture all spans

Ingest 100 percent of spans as structured logs in CloudWatch to get complete visibility. This prevents broken traces and provides you the ability to view large traces containing up to 10,000 spans for detailed insights.

Index spans as trace summaries

Index a percentage of spans as trace summaries in X-Ray to unlock end to end trace search and analytics.

Investigate transaction issues with free form analytics

Search all span attributes in the visual editor to identify the cause of issues in application transactions. This helps you answer questions about application performance and the impact end-users make based on their application transactions.

Send spans to the OpenTelemetry endpoint

Send spans to the OpenTelemetry endpoint for X-Ray traces. These spans are stored in the semantic convention format with W3C trace IDs.

> **Note**
>
> X-Ray traces automatically convert to the semantic convention format before they're stored in a log group called `aws/spans`. For more information, see The span log group.

Use CloudWatch Logs with spans

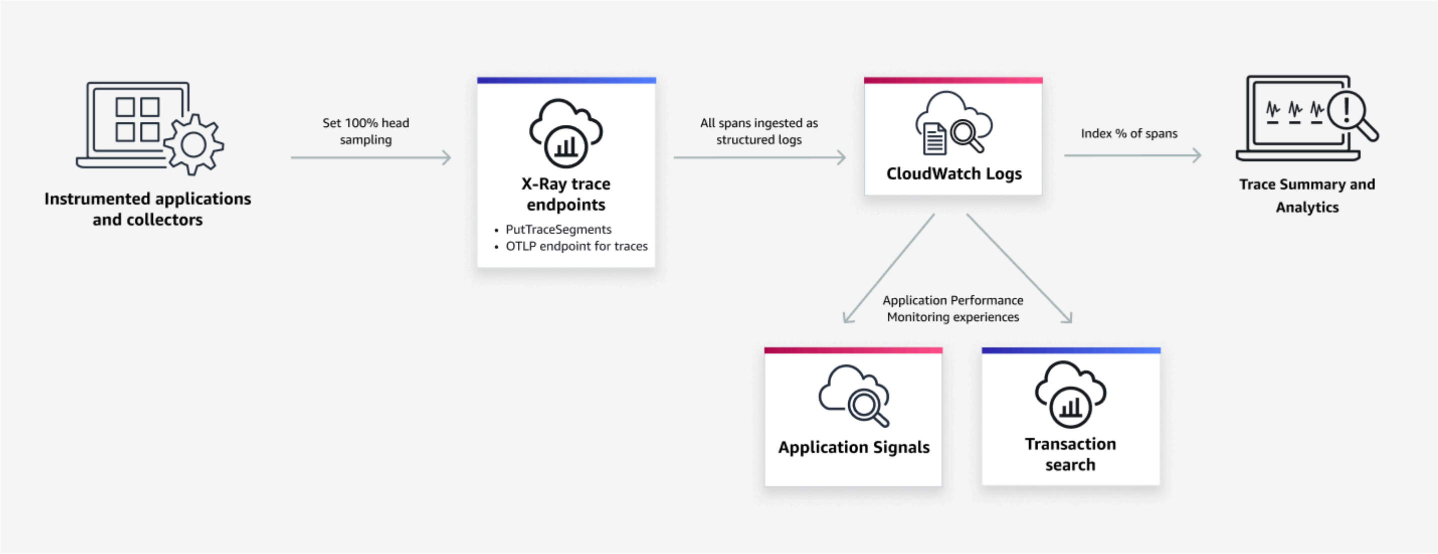
Use metric filters to extract custom metrics, subscription filters to forward data, and data masking to protect personally identifiable information.

Troubleshoot application issues

Access application dashboards, metrics, and topology when you enable Application Signals for all spans sent to CloudWatch.

How it works

When you enable Transaction Search, you unlock multiple capabilities, including features in Application Signals and CloudWatch Logs.



If you send traces to X-Ray, you can get started by enabling Transaction Search in the console or with the API. If you don't send traces to X-Ray, you can use the CloudWatch Application Signals that provides pre-packaged OpenTelemetry setup with Amazon Distro fro OpenTelemetry (ADOT), CloudWatch Agent, or use OpenTelemetry directly.

When you enable Transaction Search, spans sent to X-Ray are ingested in a log group called `aws/spans`. CloudWatch uses these spans to generate a curated application performance monitoring (APM) experience in CloudWatch Application Signals. This provides you the ability to search and analyze spans, as well as use CloudWatch Logs capabilities like anomaly and pattern detection. You can even use custom metric extraction . CloudWatch Application Signals provides you with a unified, application-centric view of your applications, services, and dependencies. It also helps you monitor and triage application health.

You can also explore spans using the interactive search and analytics experience in CloudWatch to answer any questions related to application performance or end-user impact with Transaction Search. Detect the impact on end users, find transactions in context of those issues using relevant attributes, such as customer name or order number. You can correlate transactions to business events, such as failed payments, and dive into interactions between application components to establish a root cause. With CloudWatch, you get complete application transaction coverage with correlated insights, helping you to accelerate mean time to resolution.

Pricing

For information about pricing, see Amazon CloudWatch Pricing.

Getting started with Transaction Search

If you send traces to X-Ray, you can enable Transaction Search in the CloudWatch console or with the CloudWatch API.

Topics

- Prerequisites
- Enable transaction search
- Using Transaction Search with Amazon CloudFormation

Prerequisites

Before you can enable Transaction Search, you must create a role with the following permissions.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "TransactionSearchXRayPermissions",
      "Effect": "Allow",
      "Action": [
        "xray:GetTraceSegmentDestination",
        "xray:UpdateTraceSegmentDestination",
        "xray:GetIndexingRules",
        "xray:UpdateIndexingRule"
      ],
      "Resource": "*"
    },
    {
      "Sid": "TransactionSearchLogGroupPermissions",
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutRetentionPolicy"
      ],
      "Resource": [
        "arn:aws:logs:*:*:log-group:/aws/application-signals/data:*",
        "arn:aws:logs:*:*:log-group:aws/spans:*"
      ]
    },
```

```
    {
      "Sid": "TransactionSearchLogsPermissions",
      "Effect": "Allow",
      "Action": [
        "logs:PutResourcePolicy",
        "logs:DescribeResourcePolicies"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "TransactionSearchApplicationSignalsPermissions",
      "Effect": "Allow",
      "Action": [
        "application-signals:StartDiscovery"
      ],
      "Resource": "*"
    },
    {
      "Sid": "CloudWatchApplicationSignalsCreateServiceLinkedRolePermissions",
      "Effect": "Allow",
      "Action": "iam:CreateServiceLinkedRole",
      "Resource": "arn:aws:iam::*:role/aws-service-role/application-
signals.cloudwatch.amazonaws.com/AWSServiceRoleForCloudWatchApplicationSignals",
      "Condition": {
        "StringLike": {
          "iam:AWSServiceName": "application-signals.cloudwatch.amazonaws.com"
        }
      }
    },
    {
      "Sid": "CloudWatchApplicationSignalsGetRolePermissions",
      "Effect": "Allow",
      "Action": "iam:GetRole",
      "Resource": "arn:aws:iam::*:role/aws-service-role/application-
signals.cloudwatch.amazonaws.com/AWSServiceRoleForCloudWatchApplicationSignals"
    }
  ]
}
```

> **Note**
>
> To use Transaction Search and other CloudWatch features, add the CloudWatchReadOnlyAccess policy to your role. For information about how to create a role, see IAM role creation.

Enable transaction search

You can enhance Lambda observability by using *transaction search*, which enables the capture of all trace spans for Lambda function invocation without sampling. This feature allows you to collect 100% of the spans for your functions, unaffected by the `sampled` flag in trace context propagation. This ensures that there is no additional impact to downstream dependent services. By enabling transaction search on Lambda, you gain complete visibility into your function performance and you can troubleshoot rarely occurring issues. To get started, see Transaction Search.

Enabling Transaction Search in the console

The following procedure describes how to enable Transaction Search in the console.

To enable Transaction Search in the CloudWatch console

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. From the navigation pane, under **Application Signals**, choose **Transaction Search**.
3. Choose **Enable Transaction Search**.
4. Select the box to ingest spans as structured logs, and enter a percentage of spans to be indexed. You can index spans at 1% for free and change the percentage later based on your requirements.

Enabling Transaction Search using an API

The following procedure describes how to enable Transaction Search using an API.

Step 1. Create a policy that grants access to ingest spans in CloudWatch Logs

When using the Amazon CLI or SDK to enable Transaction Search, you must configure permissions using a resource-based policy with `PutResourcePolicy`.

Example policy

The following example policy allows X-Ray to send traces to CloudWatch Logs

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "TransactionSearchXRayAccess",
            "Effect": "Allow",
            "Principal": {
                "Service": "xray.amazonaws.com"
            },
            "Action": "logs:PutLogEvents",
            "Resource": [
                "arn:partition:logs:region:account-id:log-group:aws/spans:*",
                "arn:partition:logs:region:account-id:log-group:/aws/application-
signals/data:*"
            ],
            "Condition": {
                "ArnLike": {
                    "aws:SourceArn": "arn:partition:xray:region:account-id:*"
                },
                "StringEquals": {
                    "aws:SourceAccount": "account-id"
                }
            }
        }
    ]
}
```

Example command

The following example shows how to format your Amazon CLI command with `PutResourcePolicy`.

```
aws logs put-resource-policy --policy-name MyResourcePolicy --policy-document
 '{ "Version": "2012-10-17", "Statement": [ { "Sid": "TransactionSearchXRayAccess",
 "Effect": "Allow", "Principal": { "Service": "xray.amazonaws.com" }, "Action":
 "logs:PutLogEvents", "Resource": [ "arn:partition:logs:region:account-id:log-
group:aws/spans:*", "arn:partition:logs:region:account-id:log-group:/aws/
application-signals/data:*" ], "Condition": { "ArnLike": { "aws:SourceArn":
```

```
"arn:partition:logs:region:account-id:*" }, "StringEquals": { "aws:SourceAccount":
"account-id" } } } ]}'
```

Step 2. Configure the destination of trace segments

Configure the ingestion of spans with `UpdateTraceSegmentDestination`.

Example command

The following example shows how to format your Amazon CLI command with `UpdateTraceSegmentDestination`.

```
aws xray update-trace-segment-destination --destination CloudWatchLogs
```

Step 3. Configure the amount of spans to index

Configure your desired sampling percentage with `UpdateIndexingRule`

Example command

The following example shows how to format your Amazon CLI command with `UpdateIndexingRule`.

```
aws xray update-indexing-rule --name "Default" --rule '{"Probabilistic":
 {"DesiredSamplingPercentage": number}}'
```

> **Note**
>
> After you enable Transaction Search, it can take ten minutes for spans to become available for search and analysis.

Step 4. Verify spans are available for search and analysis

To verify spans are available for search and analysis, use `GetTraceSegmentDestination`.

Example commands

The following example shows how to format your Amazon CLI command with `GetTraceSegmentDestination`.

```
aws xray get-trace-segment-destination
```

Example response

The following example shows the response you can expect when Transaction Search is active.

```
{
    "Destination": "CloudWatchLogs",
    "Status": "ACTIVE"
}
```

Using Transaction Search with Amazon CloudFormation

You can use Amazon CloudFormation to enable and configure X-Ray Transaction Search.

> **Note**
>
> To create a Amazon CloudFormation stack, see Creating your first stack .

Prerequisites

- You must have access to an Amazon account with an IAM user or role that has permissions to use Amazon EC2, Amazon S3, Amazon CloudFormation, or have administrative user access.
- You must have a Virtual Private Cloud (VPC) that has access to the internet. To keep things simple, you can use the default VPC that comes with your account. The default VPC and default subnets are sufficient for this configuration.
- Make sure Transaction Search is disabled before you enable using Amazon CDK or Amazon CloudFormation.

Enabling Transaction Search

To enable Transaction Search using CloudFormation, you need to create the following two resources.

- `AWS::Logs::ResourcePolicy`
- `AWS::XRay::TransactionSearchConfig`

1. **Create AWS::Logs::ResourcePolicy** – Create a resource policy that allows X-Ray to send traces to CloudWatch Logs

YAML

```
Resources:
  LogsResourcePolicy:
    Type: AWS::Logs::ResourcePolicy
    Properties:
      PolicyName: TransactionSearchAccess
      PolicyDocument: !Sub >
        {
          "Version": "2012-10-17",
          "Statement": [
            {
              "Sid": "TransactionSearchXRayAccess",
              "Effect": "Allow",
              "Principal": {
                "Service": "xray.amazonaws.com"
              },
              "Action": "logs:PutLogEvents",
              "Resource": [
                "arn:${AWS::Partition}:logs:${AWS::Region}:${AWS::AccountId}:log-
group:aws/spans:*",
                "arn:${AWS::Partition}:logs:${AWS::Region}:${AWS::AccountId}:log-
group:/aws/application-signals/data:*"
              ],
              "Condition": {
                "ArnLike": {
                  "aws:SourceArn": "arn:${AWS::Partition}:xray:${AWS::Region}:
${AWS::AccountId}:*"
                },
                "StringEquals": {
                  "aws:SourceAccount": "${AWS::AccountId}"
                }
              }
            }
          ]
        }
```

JSON

```
{
    "Resources": {
        "LogsResourcePolicy": {
```

```
            "Type": "AWS::Logs::ResourcePolicy",
            "Properties": {
                "PolicyName": "TransactionSearchAccess",
                "PolicyDocument": {
                    "Fn::Sub": "{\n  \"Version\": \"2012-10-17\",\n  \"Statement
\": [\n    {\n      \"Sid\": \"TransactionSearchXRayAccess\",\n      \"Effect\":
 \"Allow\",\n      \"Principal\": {\n        \"Service\": \"xray.amazonaws.com
\"\n      },\n      \"Action\": \"logs:PutLogEvents\",\n      \"Resource\":
 [\n        \"arn:${AWS::Partition}:logs:${AWS::Region}:${AWS::AccountId}:log-
group:aws/spans:*\",\n        \"arn:${AWS::Partition}:logs:${AWS::Region}:
${AWS::AccountId}:log-group:/aws/application-signals/data:*\"\n      ],\n
 \"Condition\": {\n        \"ArnLike\": {\n          \"aws:SourceArn\": \"arn:
${AWS::Partition}:xray:${AWS::Region}:${AWS::AccountId}:*\"\n        },\n
 \"StringEquals\": {\n          \"aws:SourceAccount\": \"${AWS::AccountId}\"\n
   }\n      }\n    }\n  ]\n}"
                }
            }
        }
    }
}
```

2. **Create and Configure AWS::XRay::TransactionSearchConfig** – Create the `TransactionSearchConfig` resource to enable Transaction Search.

   YAML

```
Resources:
  XRayTransactionSearchConfig:
    Type: AWS::XRay::TransactionSearchConfig
```

   JSON

```
{
  "Resources": {
    "XRayTransactionSearchConfig": {
      "Type": "AWS::XRay::TransactionSearchConfig"
    }
  }
}
```

3. (Optional) You can set the `IndexingPercentage` property to control the percentage of spans that will be indexed.

YAML

```
Resources:
  XRayTransactionSearchConfig:
    Type: AWS::XRay::TransactionSearchConfig
    Properties:
      IndexingPercentage: 50
```

JSON

```
{
  "Resources": {
    "XRayTransactionSearchConfig": {
      "Type": "AWS::XRay::TransactionSearchConfig",
      "Properties": {
        "IndexingPercentage": 20
      }
    }
  }
}
```

The IndexingPercentage value can be set between 0 and 100.

Template examples

The following example includes both the resource policy and the TransactionSearchConfig.

YAML

```
Resources:
  LogsResourcePolicy:
    Type: AWS::Logs::ResourcePolicy
    Properties:
      PolicyName: TransactionSearchAccess
      PolicyDocument: !Sub >
        {
          "Version": "2012-10-17",
          "Statement": [
            {
              "Sid": "TransactionSearchXRayAccess",
              "Effect": "Allow",
```

```
              "Principal": {
                "Service": "xray.amazonaws.com"
              },
              "Action": "logs:PutLogEvents",
              "Resource": [
                "arn:${AWS::Partition}:logs:${AWS::Region}:${AWS::AccountId}:log-
group:aws/spans:*",
                "arn:${AWS::Partition}:logs:${AWS::Region}:${AWS::AccountId}:log-
group:/aws/application-signals/data:*"
              ],
              "Condition": {
                "ArnLike": {
                  "aws:SourceArn": "arn:${AWS::Partition}:xray:${AWS::Region}:
${AWS::AccountId}:*"
                },
                "StringEquals": {
                  "aws:SourceAccount": "${AWS::AccountId}"
                }
              }
            }
          ]
        }

  XRayTransactionSearchConfig:
    Type: AWS::XRay::TransactionSearchConfig
    Properties:
      IndexingPercentage: 10
```

JSON

```
{
    "Resources": {
        "LogsResourcePolicy": {
            "Type": "AWS::Logs::ResourcePolicy",
            "Properties": {
                "PolicyName": "TransactionSearchAccess",
                "PolicyDocument": {
                    "Fn::Sub": "{\n  \"Version\": \"2012-10-17\",\n  \"Statement
\": [\n    {\n      \"Sid\": \"TransactionSearchXRayAccess\",\n      \"Effect\":
 \"Allow\",\n      \"Principal\": {\n        \"Service\": \"xray.amazonaws.com\"\n
      },\n      \"Action\": \"logs:PutLogEvents\",\n      \"Resource\": [\n
 \"arn:${AWS::Partition}:logs:${AWS::Region}:${AWS::AccountId}:log-group:aws/spans:*
\",\n        \"arn:${AWS::Partition}:logs:${AWS::Region}:${AWS::AccountId}:log-
```

```
group:/aws/application-signals/data:*\"\n      ],\n      \"Condition\": {\n
     \"ArnLike\": {\n          \"aws:SourceArn\": \"arn:${AWS::Partition}:xray:
${AWS::Region}:${AWS::AccountId}:*\"\n        },\n        \"StringEquals\": {\n
  \"aws:SourceAccount\": \"${AWS::AccountId}\"\n        }\n      }\n    }\n  ]\n}"
                }
            }
        },
        "XRayTransactionSearchConfig": {
            "Type": "AWS::XRay::TransactionSearchConfig",
            "Properties": {
                "IndexingPercentage": 20
            }
        }
    }
}
```

Here is an example using Amazon CDK in TypeScript.

CDK

```
import * as cdk from '@aws-cdk/core';
import * as logs from '@aws-cdk/aws-logs';
import * as xray from '@aws-cdk/aws-xray';

export class XRayTransactionSearchStack extends cdk.Stack {
  constructor(scope: cdk.Construct, id: string, props?: cdk.StackProps) {
    super(scope, id, props);

    // Create the resource policy
    const transactionSearchAccess = new logs.CfnResourcePolicy(this,
 'XRayLogResourcePolicy', {
      policyName: 'TransactionSearchAccess',
      policyDocument: JSON.stringify({
        Version: '2012-10-17',
        Statement: [
          {
            Sid: 'TransactionSearchXRayAccess',
            Effect: 'Allow',
            Principal: {
              Service: 'xray.amazonaws.com',
            },
            Action: 'logs:PutLogEvents',
            Resource: [
```

```
              `arn:${this.partition}:logs:${this.region}:${this.account}:log-group:aws/
spans:*`,
              `arn:${this.partition}:logs:${this.region}:${this.account}:log-group:/
aws/application-signals/data:*`,
            ],
            Condition: {
              ArnLike: {
                'aws:SourceArn': `arn:${this.partition}:xray:${this.region}:
${this.account}:*`,
              },
              StringEquals: {
                'aws:SourceAccount': this.account,
              },
            },
          },
        ],
      }),
    });

    // Create the TransactionSearchConfig with dependency
    const transactionSearchConfig = new xray.CfnTransactionSearchConfig(this,
 'XRayTransactionSearchConfig', {
      indexingPercentage: 10,
    });

    // Add the dependency to ensure Resource Policy is created first
    transactionSearchConfig.addDependsOn(transactionSearchAccess);
  }
}
```

Verifying the configuration

After deploying your Amazon CloudFormation stack, you can verify the configuration using the Amazon CLI.

aws xray get-trace-segment-destination

A successful configuration will return the following.

```
{
    "Destination": "CloudWatchLogs",
```

```
    "Status": "ACTIVE"
}
```

Spans

Spans sent to X-Ray are ingested and managed in a log group called `aws/spans`. This topic describes which CloudWatch Logs features are available for transaction spans.

Available features

The following CloudWatch Logs features are available for transaction spans.

- Metric filters – Use metric filters to extract custom metrics from spans.
- Subscriptions – Use subscriptions to access a real-time feed of span events from CloudWatch Logs.
- Log anomaly detection – Use log anomaly detection to establish a baseline for spans sent to the `aws/spans` log group.
- Contributor Insights – Use Contributor Insights to analyze span data and create a time series displaying contributor data.

Unsupported features

The following are features not supported for transaction spans.

- Spans cannot be sent to CloudWatch Logs with the `PutLogEvents` API.
- Span data cannot be enriched or transformed.

> **Note**
>
> Span ingestion is charged separately from log ingestion. For information about pricing, see Amazon CloudWatch Pricing.

Searching and analyzing spans

Transaction Search provides you with a visual editor to search and analyze all ingested spans using attributes. You can use the visual editor to narrow down transaction spans and create interactive visualizations to troubleshoot issues in your distributed applications. You can also use the CloudWatch Logs Insights query language to analyze your spans. This topic describes how to access and use the visual editor.

The visual editor

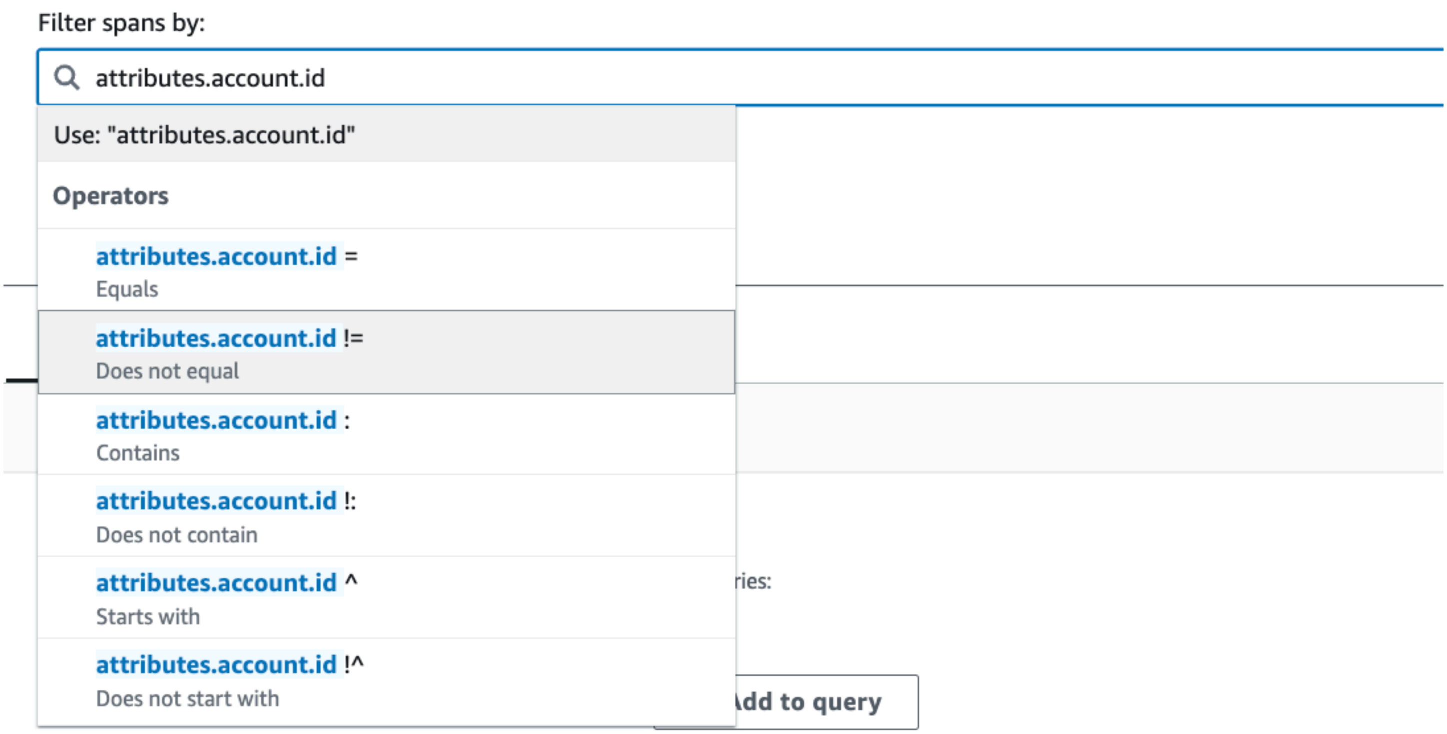
The following procedure describes how to access the visual editor.

To access the visual editor

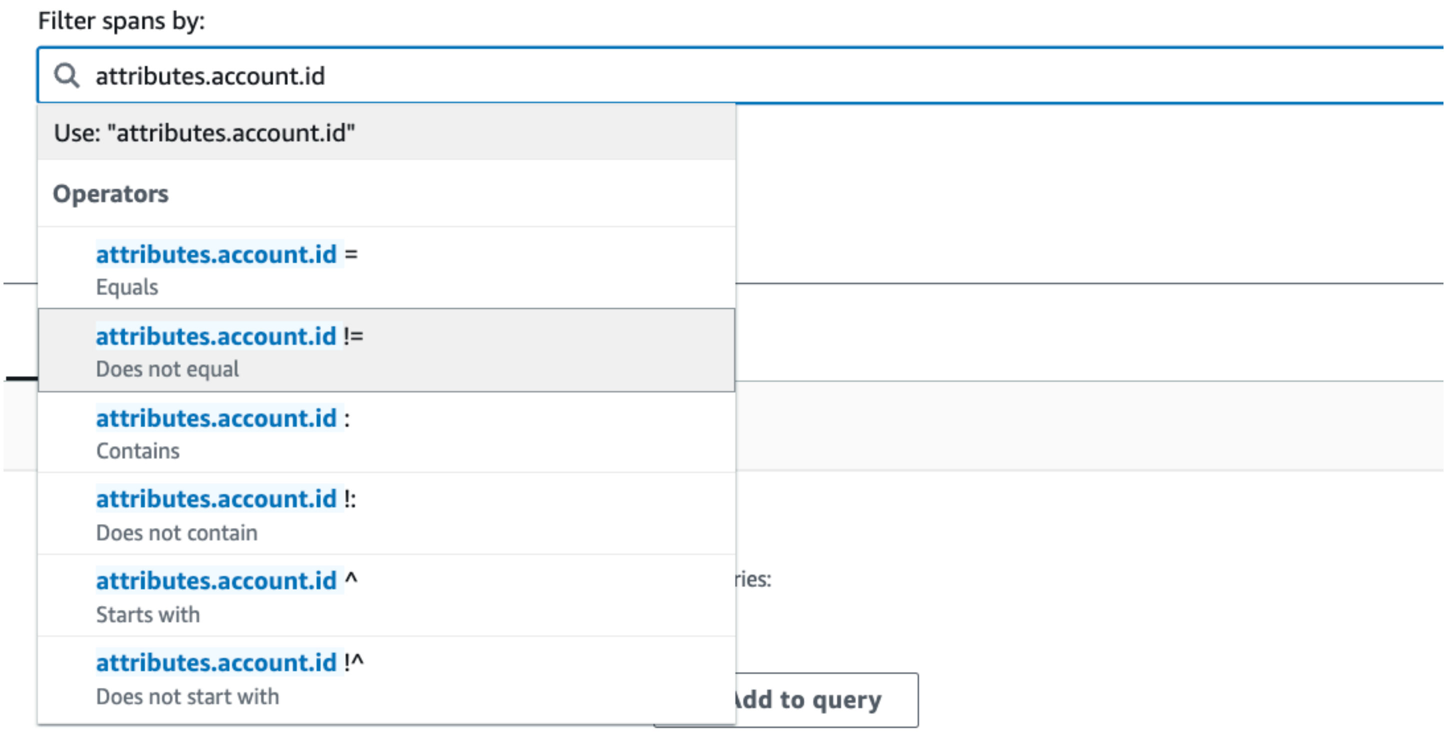
1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. From the navigation pane, choose **Application Signals**, and then choose **Transaction Search**.

Use span attributes, such as service name, span duration, and span status to narrow down transaction spans quickly. You can access these filters and more on the right side of the visual editor under **Select filters**.

This visual editor suggests a list of attributes in the span. These attributes include attributes added through auto-instrumentation and custom attributes added through custom instrumentation.



Select a span key, and enter a value to refine span results. You can filter spans using various operations, such as "Equals," "Does Not Equal," and more.



Query formats

You can run queries in the visual editor using different formats. This section describes each of these formats.

List

View spans or span events in a list format, which displays information about each span. Use this type of analysis to analyze individual spans, understand specific transactions, or identify unique patterns in transaction events. Other use cases include the following:

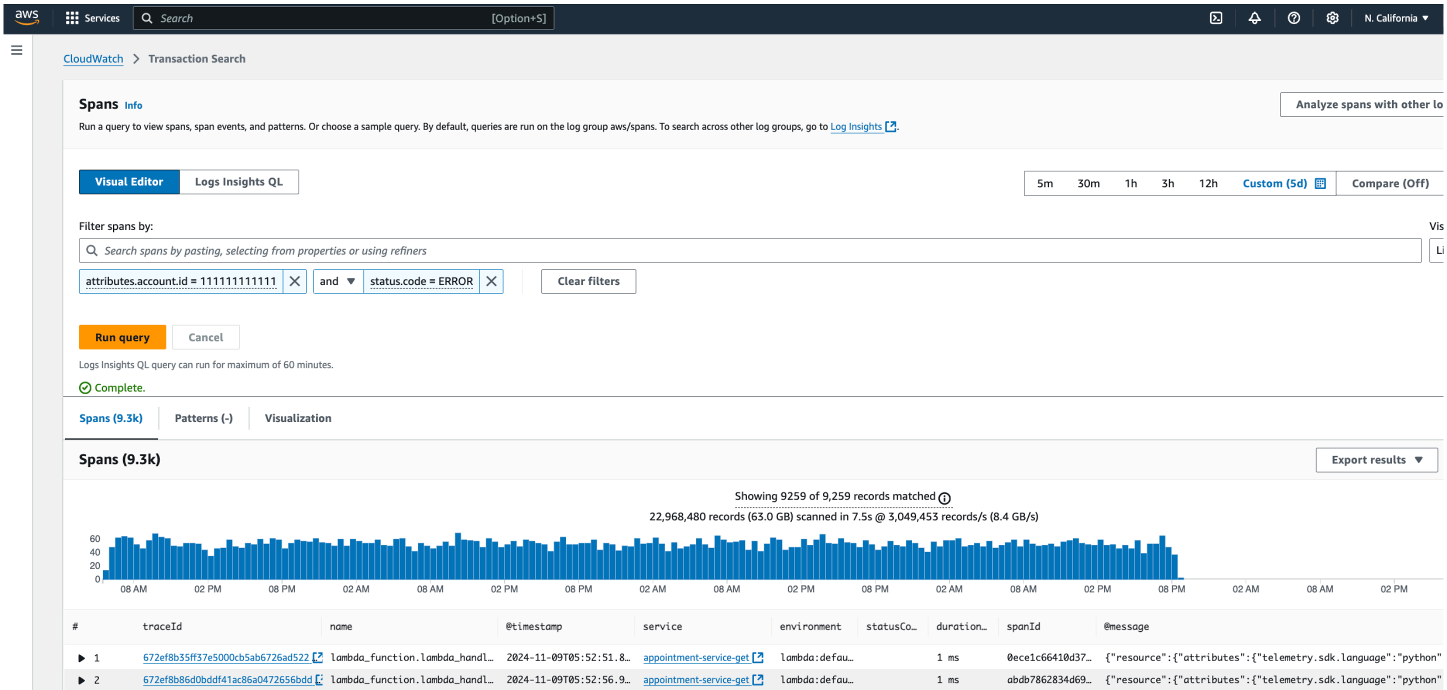
Use cases

- Troubleshoot customer support tickets
- Locate APIs or dependencies, such as database queries taking longer than 1000 milliseconds to execute
- Locate spans with errors

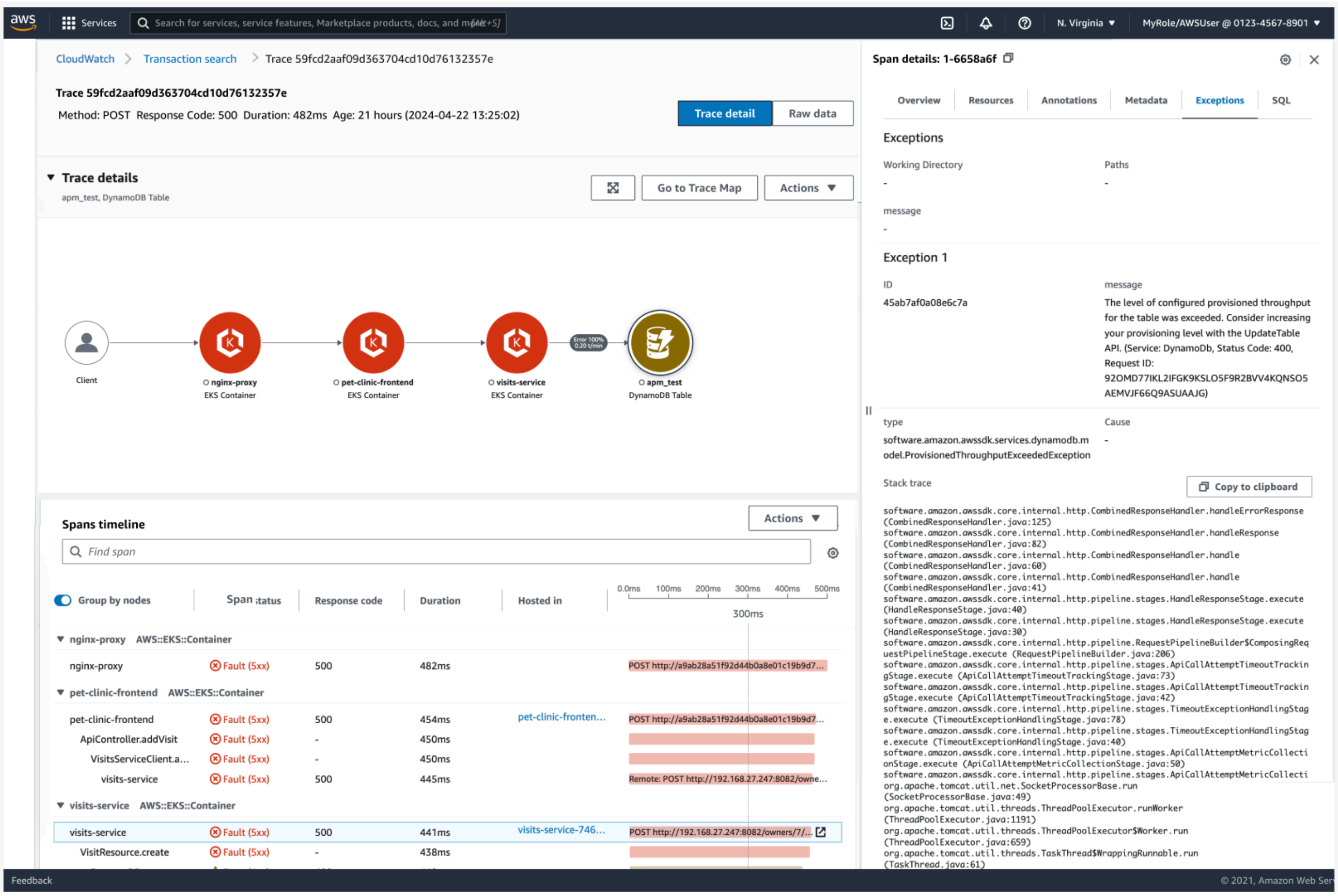
The following screenshots show how to troubleshoot a customer support ticket with this type of analysis.

Example scenario

In the visual editor, filter on all transaction spans with a particular customer issue. Before you run your query, choose **List** from the **Visualize as** dropdown.



The results show a list of spans where you can choose a trace ID to get the end-to-end journey for the transaction and determine the root cause of the issue.



Timeseries

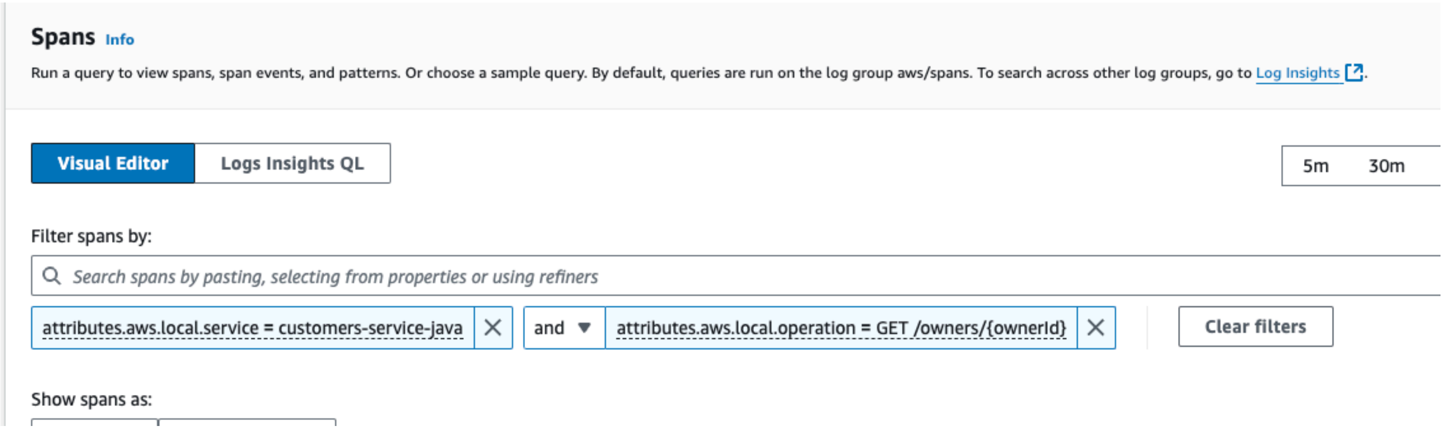
View spans or span events over time. Use this type of analysis to look at trends and spikes in transaction activity. Other use cases include the following:

- Visualize latency
- Visualize frequency of spans
- Visualize performance

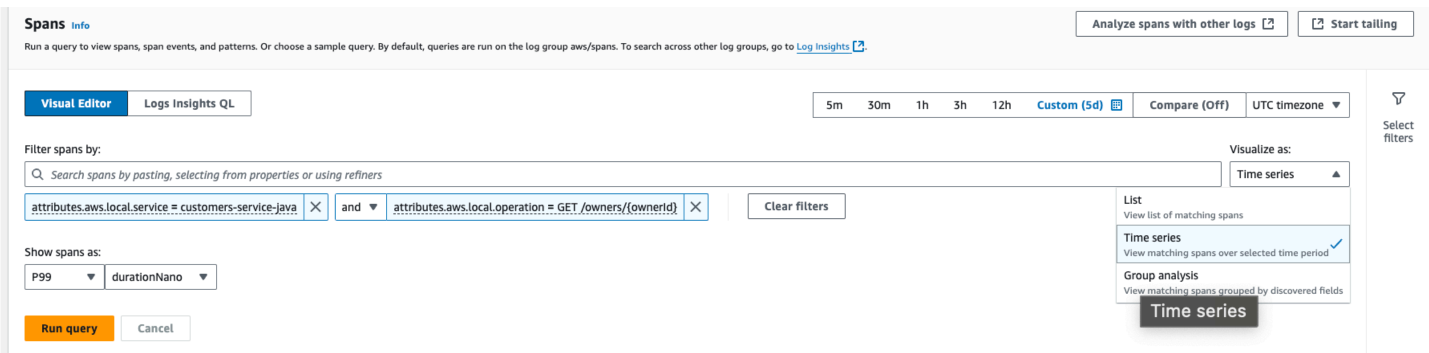
The following screenshots show how you can view p99 latency trends for an API with this type of analysis.

Example scenario

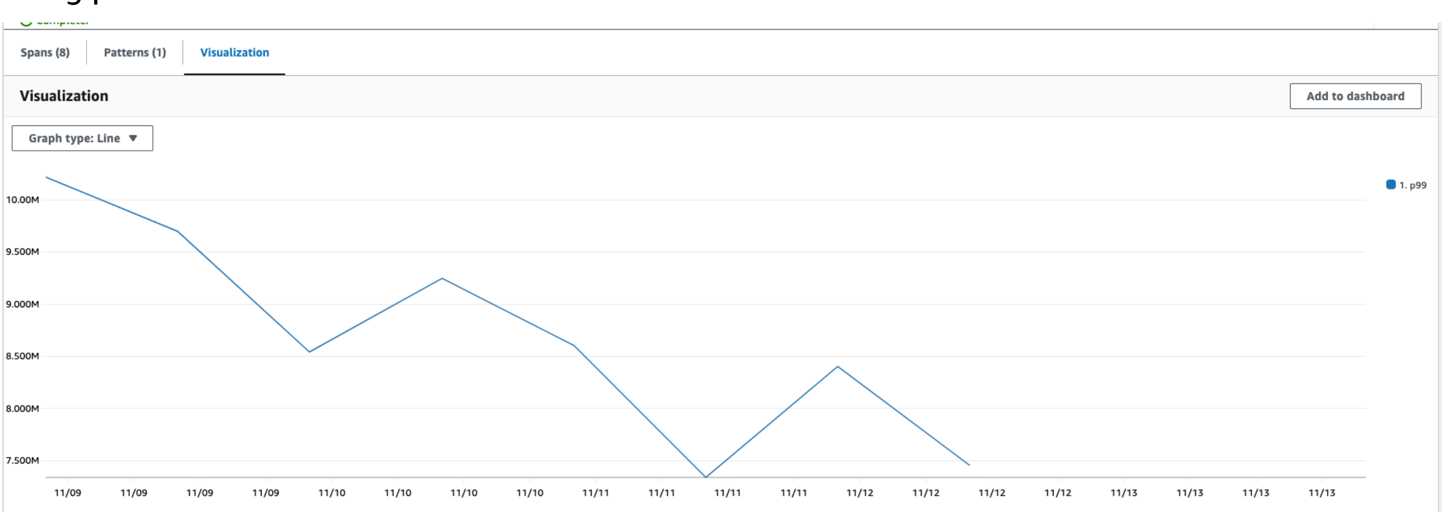
In the visual editor, filter on the service and API you want to analyze.



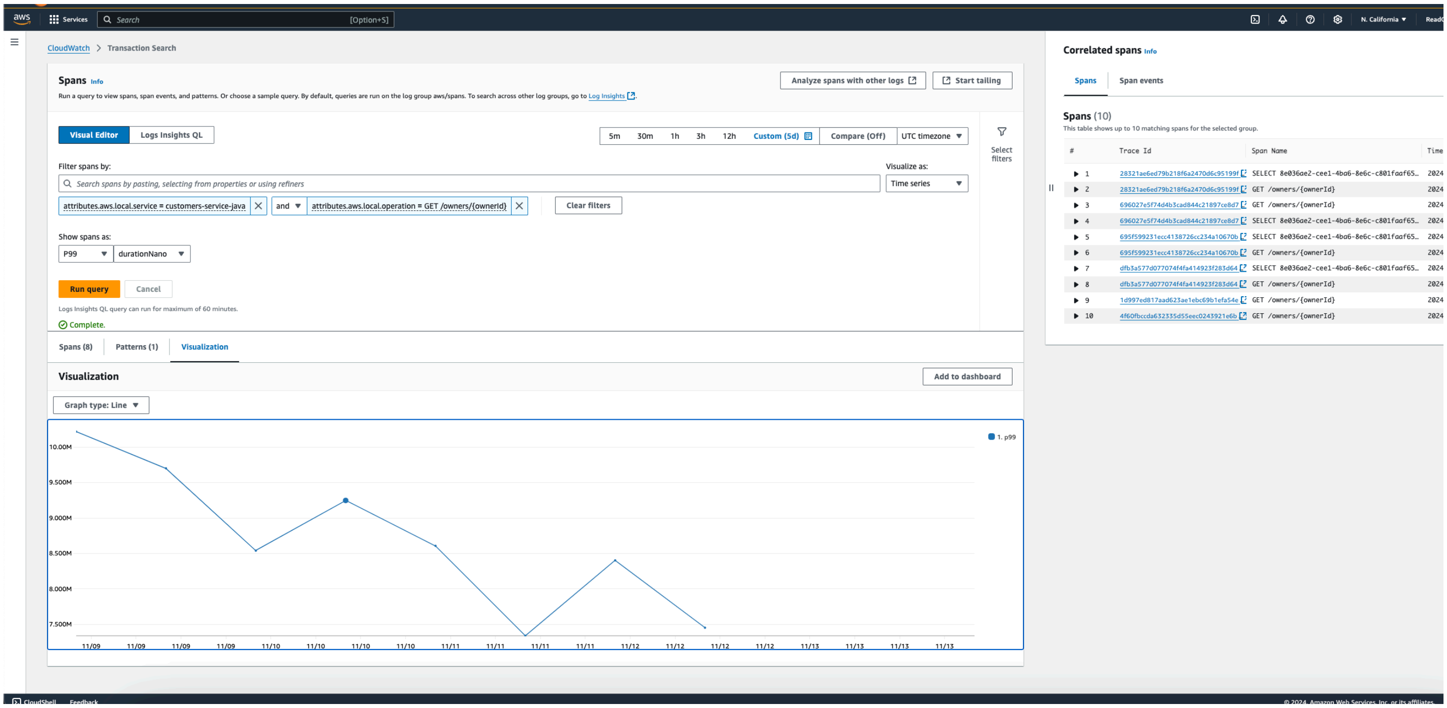
Before you run your query, choose **Time series** from the **Visualize as** dropdown. Choose **P99** for the duration statistic from the **Show span as** dropdown.



The results show a latency trend for the service, with the x-axis of the graph being time and y-axis being p99 duration.



You can choose a point on the chart to view correlated spans and span events.



Group analysis

Aggregate spans or span events based on specific attributes, such as account IDs and status codes, to display statistical metrics. Use this type of analysis to analyze spans in clusters, compare different groups, and uncover trends at the macro level. Other use cases include the following:

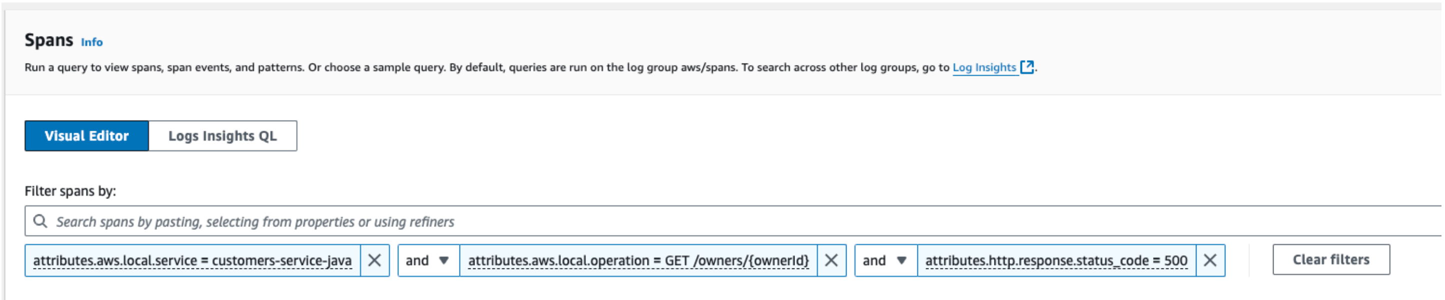
Use cases

- Identify top customers impacted by a service outage
- Identify availability zones with the most errors
- Identify the top slowest database queries

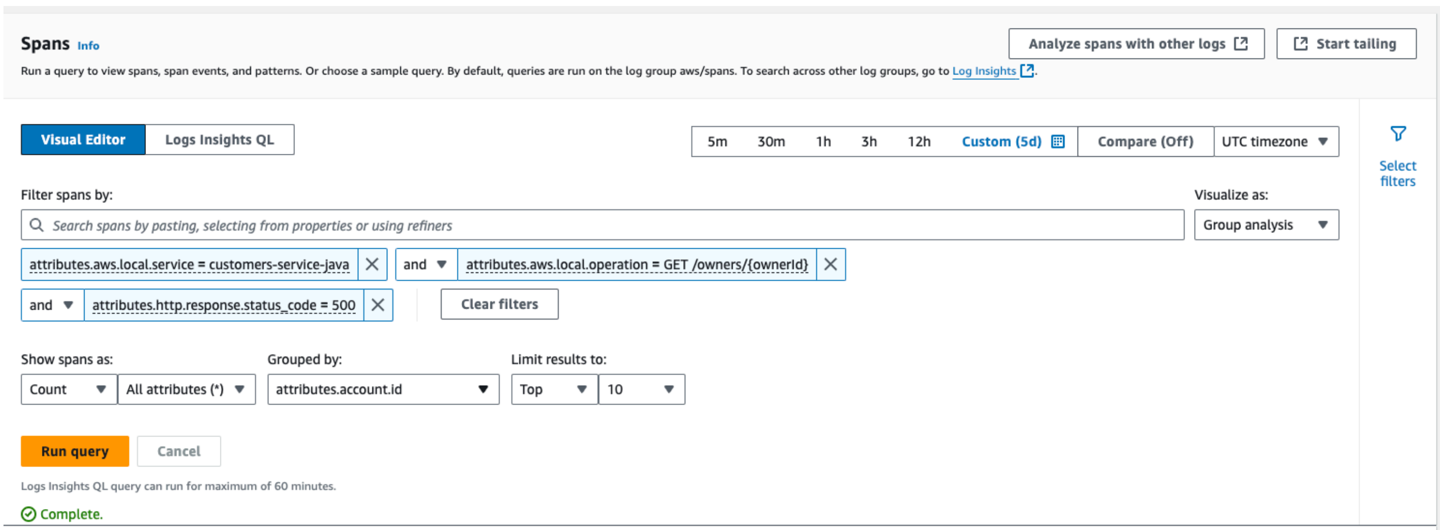
The following screenshots show how you can view the top customers impacted by a service outage with this type of analysis.

Example scenario

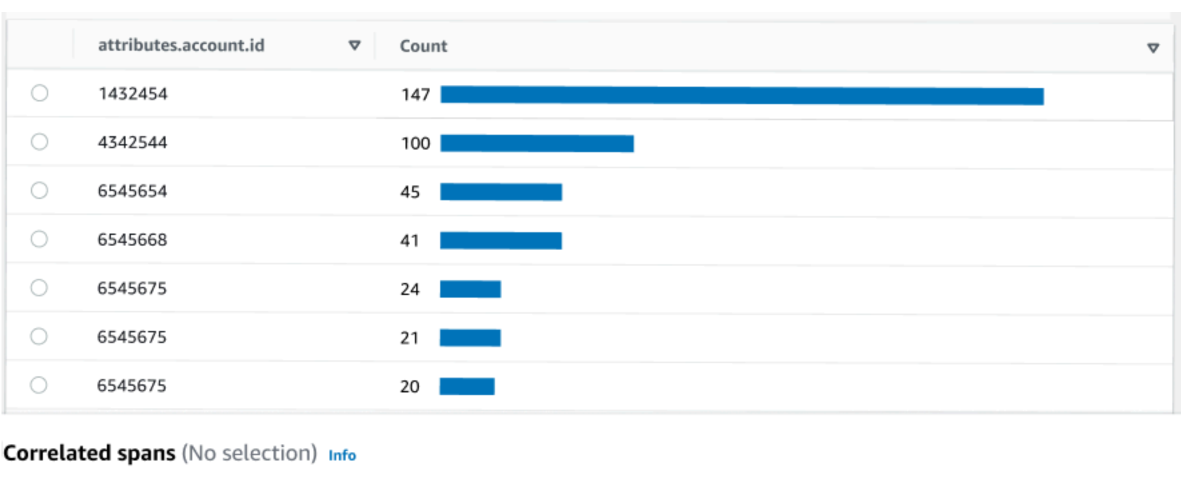
In the visual editor, you filter on the service experiencing issues.



Before you run your query, choose **Group Analysis** from the **Visualize as** dropdown. Group your query results by `account.id`, and limit the number of results to 10..



The results show the top 10 customers who experienced the most number of errors.



CloudWatch Logs Insights

You can use CloudWatch Logs Insights to analyze your spans.

Example query

The following query shows the top five slowest database queries.

```
STATS pct(durationNano, 99) as `p99` by attributes.db.statement
| SORT p99 ASC
| LIMIT 5
| DISPLAY p99,attributes.db.statement
```

Example query

The following query shows which top five services are throwing errors.

```
FILTER `attributes.http.response.status_code` >= 500
| STATS count(*) as `count` by attributes.aws.local.service as service
| SORT count ASC
| LIMIT 5
| DISPLAY count,service
```

Ingesting spans for complete visibility

Recording all transaction spans provides comprehensive visibility into application issues. It enables you to debug customer support tickets or troubleshoot rarely occurring p99 API latency spikes, which is crucial when identifying the root cause of issues in customer-facing and mission-critical applications. You can create a cost-effective strategy to start capturing 100% of trace spans in CloudWatch by configuring the head sampling rate and then adjusting a lower span indexing rate.

Setting up head sampling

Head sampling is a tracing technique capturing requests at the beginning of a trace, which is based on a set rate or condition.

When the head sampling rate is set to 100%, it captures the beginning of every trace without skips, guaranteeing complete visibility into all incoming requests, and that no transaction data is missed.

You can configure head sampling if you're using X-Ray or Amazon Distro for OpenTelemetry SDKs or the OpenTelemetry SDK.

If you're using X-Ray or Amazon Distro for OpenTelemetry SDKs

Navigate to your sampling rules in the console, and set the fixed sampling rate to 100%. This guarantees all trace spans are captured and ingested into CloudWatch logs. For more information, see Configuring sampling rules

If you're using the OpenTelemetry SDK

To record 100% of spans and get complete visibility, set your sampling configuration to `always_on`. For more information, see Language APIs & SDKs on the OpenTelemetry website.

Features unlocked with head sampling

When you enable Transaction Search, all spans collected from your application through head sampling are ingested as structured logs in CloudWatch. This provides you with the following features:

- The ability to search span attributes and analyze span events in a visual editor.
- The ability to visualize traces containing up to 10,000 spans.
- Total support for OpenTelemetry, which includes the ability to embed business events into spans for analysis and use span links to define connections between traces for end-to-end viewing.
- Access to application dashboards, metrics, and topology with CloudWatch Application Signals enabled for all spans sent to CloudWatch.

> **Note**
>
> Because spans are available in a log group called `aws/spans`, you can use CloudWatch Logs features with transaction spans. For more information, see The span log group.

Setting up span indexing with trace summaries

Trace summaries can help you debug transactions and are beneficial for asynchronous processes. You only need to index a small percentage of spans as trace summaries.

You configure span indexing when you enable Transaction Search in the console or with the API. To enable Transaction Search, see Getting started with Transaction Search.

Features unlocked with trace summaries

The key features of X-Ray trace summaries include the following:

- Trace summary search – Search and find traces from trace summaries.
- Trace summary analytics – Interpret trace data.
- Trace insights – Analyze trace data to identify application issues.

Monitoring spans across accounts

Spans sent to X-Ray are ingested and managed in a log group called `aws/spans`. To monitor spans across multiple accounts, you must enable Transaction Search across all source and monitoring accounts and enable cross-account observability for logs and traces. When you enable cross-account observability, you can search up to 10,000 accounts and get visibility into traces across accounts. This feature is provided at no extra cost for the `aws/spans` log group. If you enable cross-account observability for trace summaries, the first trace summary copy is free. For more information about pricing, see Amazon CloudWatch Pricing.

Adding custom attributes

CloudWatch Application Signals utilizes OpenTelemetry to auto-instrument your applications and collect spans from popular libraries in different languages, such as Java, Python, and more.

Auto-instrumentation captures information, such as database queries, HTTP requests, cache accesses, and external service calls, which allows you to troubleshoot application performance issues.

You can add custom instrumentation to enrich spans with business-specific data or other information you wish to capture. This data can be recorded as a custom attribute or a span event, providing insights tailored to your troubleshooting needs.

> **Note**
>
> For information about adding custom attributes or span events in a different language, see Language APIs and SDKS in the *OpenTelemetry website*.

Custom attributes

You can add business related attributes or any other attributes to your spans in all languages OpenTelemetry supports. The following is a Java code snippet that adds an order id and customer details to a span.

```
import io.opentelemetry.api.trace.Span;

public class OrderProcessor {

    public void processOrder() {
        Span span = Span.current();
        span.setAttribute("order.id", "123456");
        span.setAttribute("customer.name", "John Doe");
        span.setAttribute("customer.id", "4343dfdd");

        // Your order processing logic here
        System.out.println("Order processed with custom attributes");
    }
}
```

After these attributes have been added to the span, they become available to search and analyze in the Transaction Search visual editor.

Span events

A span event is typically used to denote a meaningful, singular point in time during a span duration. Exceptions are auto-captured as span events through auto-instrumentation, but you can also add custom business events, such as payment-status or cart-abandonment. For more information, see Span events on the OpenTelemetry website.

You can embed span events to your spans in all the languages CloudWatch Application Signals and OpenTelemetry support. The following is a Java code snippet of adding a custom event to a span.

```
import io.opentelemetry.api.trace.Span;

public class OrderProcessor {

    public void bookOrder() {
        Span span = Span.current();
```

```
        // Add a booking started event
        span.addEvent("booking started");

        // Add a payment succeeded event or failed event
        span.addEvent("booking failed");
    }
}
```

Prerequisites for the CloudWatch agent

When using the CloudWatch agent to emit span events to X-Ray, you must turn on the `` `transit_spans_in_otlp_format` `` flag in your configuration.

```
{
  "traces": {
    ...
    "transit_spans_in_otlp_format": true
    ...
  }
}
```

After you add these events, they become available in the Transaction Search visual editor.

CloudWatch Logs queries

You can query span events in CloudWatch Logs to view advanced insights. The following example query commands show how to analyze exceptions thrown by your application:

```
fields jsonparse(@message) as js
| unnest js.events into event
| filter event.name = "exception"
| display event.attributes.`exception.stacktrace`
```

```
fields jsonparse(@message) as js
| unnest js.events into event
| filter event.name = "exception"
| stats count() by event.attributes.`exception.type`
```

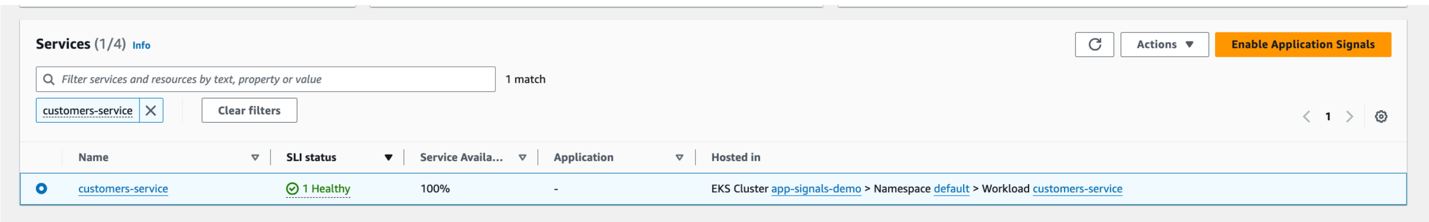
Troubleshooting application issues

With Application Signals, you can troubleshoot rarely occurring latency spikes in your applications. After you enable Transaction Search and configure a head sampling rate capturing 100% of spans, you get complete visibility into any application issue. The following scenario describes how Application Signals can be used with transaction spans to monitor your services and identify service quality issues.

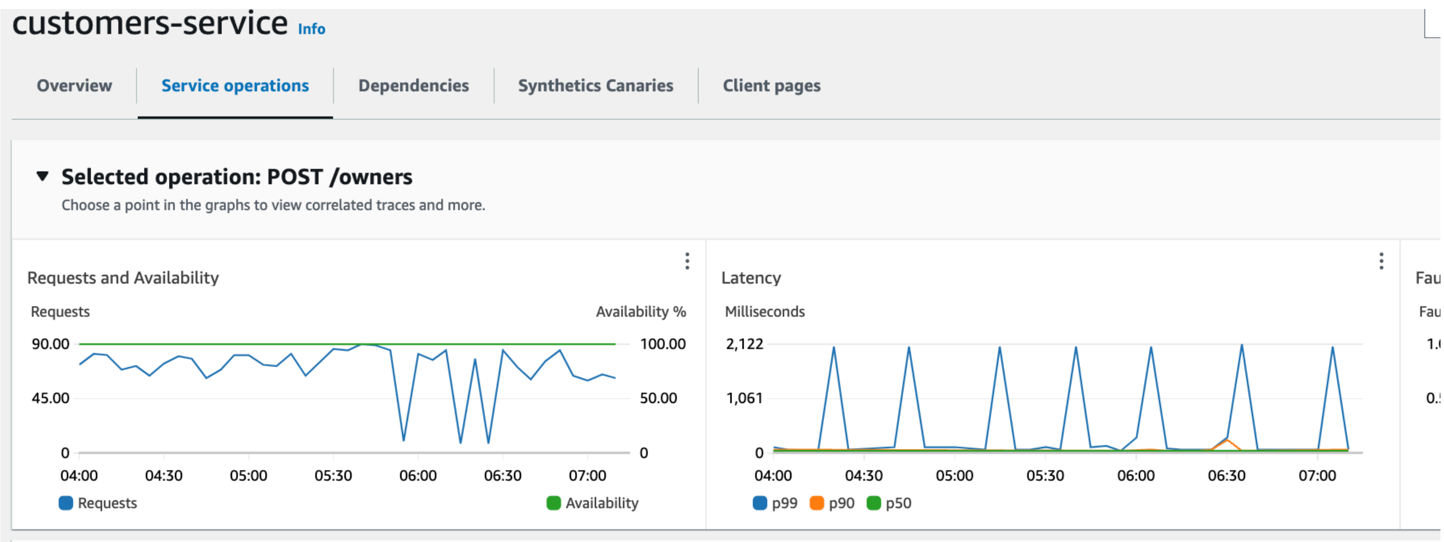
Example troubleshooting scenario

This scenario focuses on a pet clinic application composed of several micro-services calling third-party payment APIs. These calls have been intermittently slow, thus impacting revenue.

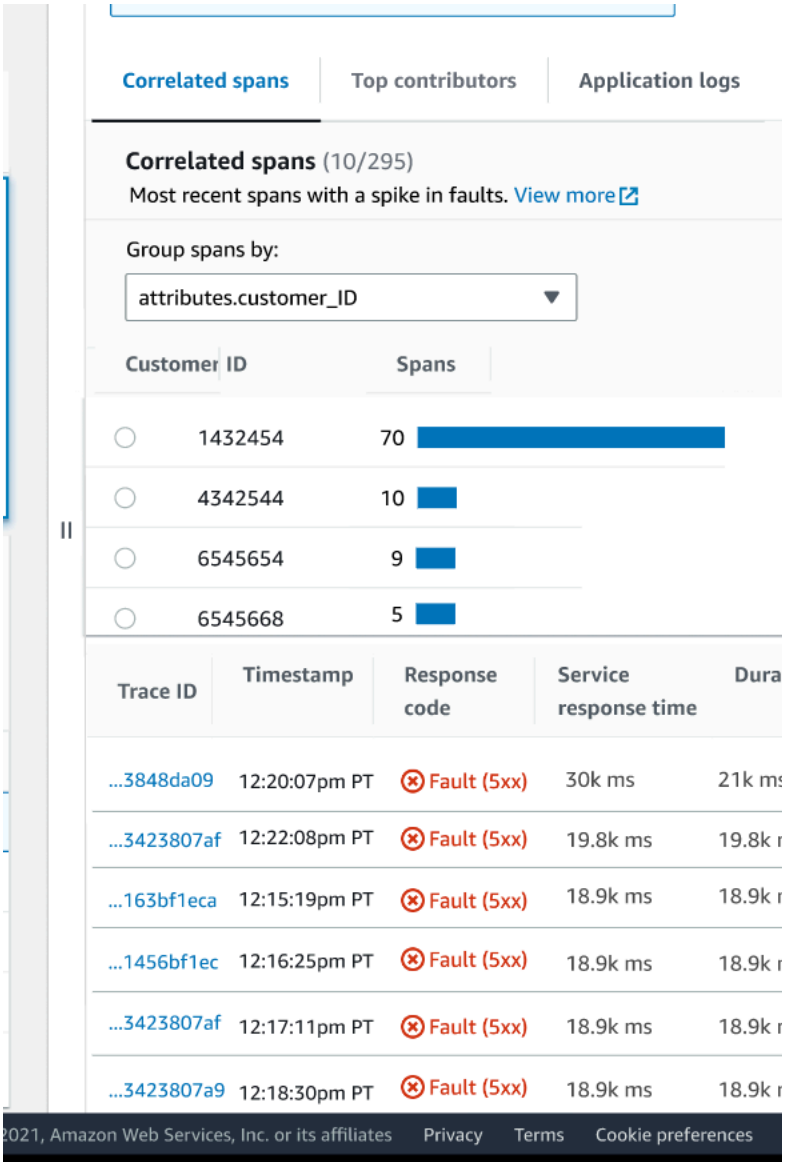
Jane opens the CloudWatch Application Signals console and notices a customer-service application responsible for registering customers is healthy and not breaching any SLOs.



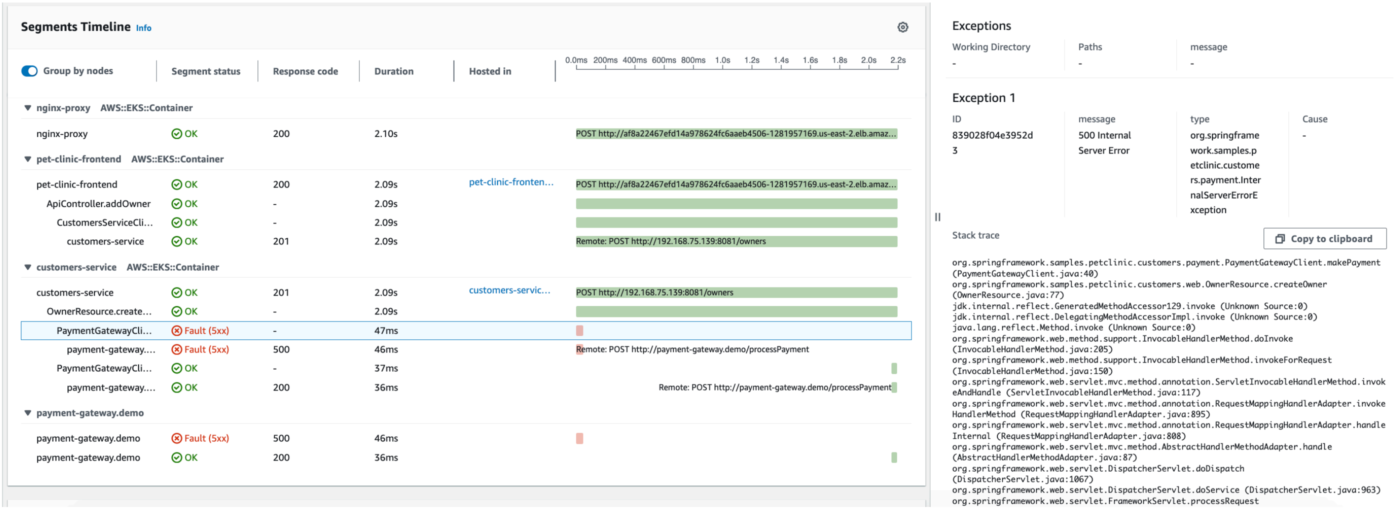
She opens the service to investigate any patterns of rarely occurring failures and notices the registration API experienced intermittent p99 latency spikes.



Jane chooses a datapoint in the latency chart to view correlated spans. She groups the spans by customer ID to view all the customers who are impacted by the latency spikes.



Jane selects one of the correlated spans with a fault status, which opens the trace detail page for the selected trace. She scrolls to the segments timeline section to follow the call path, where she notices that calls to the payment gateway have been failing and preventing customers from registering.



Synthetic monitoring (canaries)

You can use Amazon CloudWatch Synthetics to create *canaries*, configurable scripts that run on a schedule, to monitor your endpoints and APIs. Canaries follow the same routes and perform the same actions as a customer, which makes it possible for you to continually verify your customer experience even when you don't have any customer traffic on your applications. By using canaries, you can discover issues before your customers do.

Canaries are scripts written in Node.js or Python. They create Lambda functions in your account that use Node.js or Python as a framework. Canaries work over both HTTP and HTTPS protocols. Canaries use Lambda layers that contain the CloudWatch Synthetics library. The library contains the NodeJS version of CloudWatch Synthetics for NodeJS canaries and the Python version of CloudWatch Synthetics for Python canaries. The layers belong to the CloudWatch Synthetics service account. Libraries never transmit or store customer information. All customer data is stored in the customer account only.

Canaries offer programmatic access to a headless Google Chrome Browser via Puppeteer or Selenium Webdriver. For more information about Puppeteer, see [Puppeteer](). For more information about Selenium, see [www.selenium.dev/]().

Canaries check the availability and latency of your endpoints and can store load time data and screenshots of the UI. They monitor your REST APIs, URLs, and website content, and they can check for unauthorized changes from phishing, code injection and cross-site scripting.

CloudWatch Synthetics is integrated with [Application Signals](), which can discover and monitor your application services, clients, Synthetics canaries, and service dependencies. Use Application

Signals to see a list or visual map of your services, view health metrics based on your service level objectives (SLOs), and drill down to see correlated X-Ray traces for more detailed troubleshooting. To see your canaries in Application Signals, turn on X-Ray active tracing. Your canaries are displayed on the Service Map connected to your services, and in the Service detail page of the services they call.

You can run a canary once or on a regular schedule. Canaries can run as often as once per minute. You can use both cron and rate expressions to schedule canaries.

For information about security issues to consider before you create and run canaries, see Security considerations for Synthetics canaries.

By default, canaries create several CloudWatch metrics in the `CloudWatchSynthetics` namespace. These metrics have `CanaryName` as a dimension. Canaries that use the `executeStep()` or `executeHttpStep()` function from the function library also have `StepName` as a dimension. For more information about the canary function library, see Library functions available for canary scripts.

CloudWatch Synthetics integrates well with the X-Ray Trace Map, which uses CloudWatch with Amazon X-Ray to provide an end-to-end view of your services to help you more efficiently pinpoint performance bottlenecks and identify impacted users. Canaries that you create with CloudWatch Synthetics appear on the trace map. For more information, see X-Ray Trace Map.

CloudWatch Synthetics is currently available in all commercial Amazon Regions and the GovCloud Regions.

> **Note**
>
> In Asia Pacific (Osaka), Amazon PrivateLink is not supported. In Asia Pacific (Jakarta), Amazon PrivateLink and X-Ray are not supported.

Topics

Required roles and permissions for CloudWatch canaries

Both the users who create and manage canaries, and the canaries themselves, must have certain permissions.

Required roles and permissions for users who manage CloudWatch canaries

To view canary details and the results of canary runs, you must be signed in as a user with either the `CloudWatchSyntheticsFullAccess` or the `CloudWatchSyntheticsReadOnlyAccess` policies attached. To read all Synthetics data in the console, you also need the `AmazonS3ReadOnlyAccess` and `CloudWatchReadOnlyAccess` policies. To view the source code used by canaries, you also need the `AWSLambda_ReadOnlyAccess` policy.

To create canaries, you must be signed in as an user who has the `CloudWatchSyntheticsFullAccess` policy or a similar set of permissions. To create IAM roles for the canaries, you also need the following inline policy statement:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iam:CreateRole",
```

```
                "iam:CreatePolicy",
                "iam:AttachRolePolicy"
            ],
            "Resource": [
                "arn:aws:iam::*:role/service-role/CloudWatchSyntheticsRole*",
                "arn:aws:iam::*:policy/service-role/CloudWatchSyntheticsPolicy*"
            ]
        }
    ]
}
```

> **Important**
>
> Granting a user the `iam:CreateRole`, `iam:CreatePolicy`, and `iam:AttachRolePolicy` permissions gives that user full administrative access to your Amazon account. For example, a user with these permissions can create a policy that has full permissions for all resources and can attach that policy to any role. Be very careful about who you grant these permissions to.

For information about attaching policies and granting permissions to users, see Changing Permissions for an IAM User and To embed an inline policy for a user or role.

Required roles and permissions for canaries

Each canary must be associated with an IAM role that has certain permissions attached. When you create a canary using the CloudWatch console, you can choose for CloudWatch Synthetics to create an IAM role for the canary. If you do, the role will have the permissions needed.

If you want to create the IAM role yourself, or create an IAM role that you can use when using the Amazon CLI or APIs to create a canary, the role must contain the permissions listed in this section.

All IAM roles for canaries must include the following trust policy statement.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "lambda.amazonaws.com"
```

```
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

Additionally, the canary's IAM role needs one of the following statements.

Basic canary that doesn't use Amazon KMS or need Amazon VPC access

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::path/to/your/s3/bucket/canary/results/folder"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketLocation"
            ],
            "Resource": [
                "arn:aws:s3:::name/of/the/s3/bucket/that/contains/canary/results"

            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "logs:CreateLogGroup"
            ],
            "Resource": [
                "arn:aws:logs:canary_region_name:canary_account_id:log-group:/aws/
lambda/cwsyn-canary_name-*"
            ]
```

```
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:ListAllMyBuckets",
                "xray:PutTraceSegments"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Resource": "*",
            "Action": "cloudwatch:PutMetricData",
            "Condition": {
               "StringEquals": {
                "cloudwatch:namespace": "CloudWatchSynthetics"
               }
            }
        }
    ]
}
```

Canary that uses Amazon KMS to encrypt canary artifacts but does not need Amazon VPC access

```
{
    "Version": "2012-10-17",
    "Statement": [{
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::path/to/your/S3/bucket/canary/results/folder"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketLocation"
```

```
            ],
            "Resource": [
                "arn:aws:s3:::name/of/the/S3/bucket/that/contains/canary/results"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "logs:CreateLogGroup"
            ],
            "Resource": [
                "arn:aws:logs:canary_region_name:canary_account_id:log-group:/aws/
lambda/cwsyn-canary_name-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:ListAllMyBuckets",
                "xray:PutTraceSegments"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Resource": "*",
            "Action": "cloudwatch:PutMetricData",
            "Condition": {
                "StringEquals": {
                    "cloudwatch:namespace": "CloudWatchSynthetics"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt",
                "kms:GenerateDataKey"
            ],
```

```
            "Resource":
 "arn:aws:kms:KMS_key_region_name:KMS_key_account_id:key/KMS_key_id",
            "Condition": {
                "StringEquals": {
                    "kms:ViaService": [
                        "s3.region_name_of_the_canary_results_S3_bucket.amazonaws.com"
                    ]
                }
            }
        }
    ]
}
```

Canary that does not use Amazon KMS but does need Amazon VPC access

```
{
    "Version": "2012-10-17",
    "Statement": [{
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::path/to/your/S3/bucket/canary/results/folder"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketLocation"
            ],
            "Resource": [
                "arn:aws:s3:::name/of/the/S3/bucket/that/contains/canary/results"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "logs:CreateLogGroup"
            ],
```

```
            "Resource": [
                "arn:aws:logs:canary_region_name:canary_account_id:log-group:/aws/
lambda/cwsyn-canary_name-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:ListAllMyBuckets",
                "xray:PutTraceSegments"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Resource": "*",
            "Action": "cloudwatch:PutMetricData",
            "Condition": {
                "StringEquals": {
                    "cloudwatch:namespace": "CloudWatchSynthetics"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterfaces",
                "ec2:DeleteNetworkInterface"
            ],
            "Resource": [
                "*"
            ]
       }
    ]
}
```

Canary that uses Amazon KMS to encrypt canary artifacts and also needs Amazon VPC access

If you update a non-VPC canary to start using a VPC, you'll need to update the canary's role to include the network interface permissions listed in the following policy.

```
{
    "Version": "2012-10-17",
    "Statement": [{
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::path/to/your/S3/bucket/canary/results/folder"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketLocation"
            ],
            "Resource": [
                "arn:aws:s3:::name/of/the/S3/bucket/that/contains/canary/results"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "logs:CreateLogGroup"
            ],
            "Resource": [
                "arn:aws:logs:canary_region_name:canary_account_id:log-group:/aws/
lambda/cwsyn-canary_name-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:ListAllMyBuckets",
                "xray:PutTraceSegments"
            ],
            "Resource": [
                "*"
            ]
        },
```

```
        {
            "Effect": "Allow",
            "Resource": "*",
            "Action": "cloudwatch:PutMetricData",
            "Condition": {
                "StringEquals": {
                    "cloudwatch:namespace": "CloudWatchSynthetics"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterfaces",
                "ec2:DeleteNetworkInterface"
            ],
            "Resource": [
                "*"
            ]
       },
       {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt",
                "kms:GenerateDataKey"
            ],
            "Resource":
 "arn:aws:kms:KMS_key_region_name:KMS_key_account_id:key/KMS_key_id",
            "Condition": {
                "StringEquals": {
                    "kms:ViaService": [
                        "s3.region_name_of_the_canary_results_S3_bucket.amazonaws.com"
                    ]
                }
            }
        }
    ]
}
```

Amazon managed policies for CloudWatch Synthetics

To add permissions to users, groups, and roles, it is easier to use Amazon managed policies than to write policies yourself. It takes time and expertise to create IAM customer managed policies that provide your team with only the permissions they need. To get started quickly, you can use our Amazon managed policies. These policies cover common use cases and are available in your Amazon account. For more information about Amazon managed policies, see Amazon managed policies Amazon managed policies in the IAM User Guide.

Amazon services maintain and update Amazon managed policies. You can't change the permissions in Amazon managed policies. Services occasionally change the permissions in an Amazon managed policy. This type of update affects all identities (users, groups, and roles) where the policy is attached.

CloudWatch Synthetics updates to Amazon managed policies

View details about updates to Amazon managed policies for CloudWatch Synthetics since this service began tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the CloudWatch Document history page.

Change	Description	Date
Redundant actions removed from **CloudWatchSyntheticsFullAccess**	CloudWatch Synthetics removed the `s3:PutBucketEncryption` and `lambda:GetLayerVersionByArn` actions from **CloudWatchSyntheticsFullAccess** policy because those actions were redundant with other permissions in the policy. The removed actions did not provide any permissions, and there's no net change to the permissions granted by the policy.	March 12, 2021

Change	Description	Date
CloudWatch Synthetics started tracking changes	CloudWatch Synthetics started tracking changes for its Amazon managed policies.	March 10, 2021

CloudWatchSyntheticsFullAccess

Here are the contents of the `CloudWatchSyntheticsFullAccess` policy:

```
{
    "Version": "2012-10-17",
    "Statement": [{
            "Effect": "Allow",
            "Action": [
                "synthetics:*"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:CreateBucket",
                "s3:PutEncryptionConfiguration"
            ],
            "Resource": [
                "arn:aws:s3:::cw-syn-results-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:ListRoles",
                "s3:ListAllMyBuckets",
                "xray:GetTraceSummaries",
                "xray:BatchGetTraces",
                "apigateway:GET"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
```

```
            "Action": [
                "s3:GetBucketLocation"
            ],
            "Resource": "arn:aws:s3:::*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:ListBucket"
            ],
            "Resource": "arn:aws:s3:::cw-syn-*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObjectVersion"
            ],
            "Resource": "arn:aws:s3:::aws-synthetics-library-*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": [
                "arn:aws:iam::*:role/service-role/CloudWatchSyntheticsRole*"
            ],
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": [
                        "lambda.amazonaws.com",
                        "synthetics.amazonaws.com"
                    ]
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:GetRole",
                "iam:ListAttachedRolePolicies"
            ],
            "Resource": [
```

```
                "arn:aws:iam::*:role/service-role/CloudWatchSyntheticsRole*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:GetMetricData",
                "cloudwatch:GetMetricStatistics"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:PutMetricAlarm",
                "cloudwatch:DeleteAlarms"
            ],
            "Resource": [
                "arn:aws:cloudwatch:*:*:alarm:Synthetics-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:DescribeAlarms"
            ],
            "Resource": [
                "arn:aws:cloudwatch:*:*:alarm:*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:GetLogRecord",
                "logs:DescribeLogStreams",
                "logs:StartQuery",
                "logs:GetLogEvents",
                "logs:FilterLogEvents",
                "logs:GetLogGroupFields"
            ],
            "Resource": [
                "arn:aws:logs:*:*:log-group:/aws/lambda/cwsyn-*"
            ],
            "Condition": {
```

```
                "StringEquals": {
                    "aws:ResourceAccount": "${aws:PrincipalAccount}"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "lambda:CreateFunction",
                "lambda:AddPermission",
                "lambda:PublishVersion",
                "lambda:UpdateFunctionCode",
                "lambda:UpdateFunctionConfiguration",
                "lambda:GetFunctionConfiguration",
                "lambda:GetFunction",
                "lambda:DeleteFunction",
                "lambda:ListTags",
                "lambda:TagResource",
                "lambda:UntagResource"
            ],
            "Resource": [
                "arn:aws:lambda:*:*:function:cwsyn-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "lambda:GetLayerVersion",
                "lambda:PublishLayerVersion",
                "lambda:DeleteLayerVersion"
            ],
            "Resource": [
                "arn:aws:lambda:*:*:layer:cwsyn-*",
                "arn:aws:lambda:*:*:layer:Synthetics:*",
                "arn:aws:lambda:*:*:layer:Synthetics_Selenium:*",
                "arn:aws:lambda:*:*:layer:AWS-CW-Synthetics*:*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeVpcs",
                "ec2:DescribeSubnets",
                "ec2:DescribeSecurityGroups"
```

```
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "sns:ListTopics"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "sns:CreateTopic",
                "sns:Subscribe",
                "sns:ListSubscriptionsByTopic"
            ],
            "Resource": [
                "arn:*:sns:*:*:Synthetics-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "kms:ListAliases"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "kms:DescribeKey"
            ],
            "Resource": "arn:aws:kms:*:*:key/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt"
            ],
```

```
            "Resource": "arn:aws:kms:*:*:key/*",
            "Condition": {
                "StringLike": {
                    "kms:ViaService": [
                        "s3.*.amazonaws.com"
                    ]
                }
            }
        }
    ]
}
```

CloudWatchSyntheticsReadOnlyAccess

Here are the contents of the `CloudWatchSyntheticsReadOnlyAccess` policy:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "synthetics:Describe*",
                "synthetics:Get*",
                "synthetics:List*"
            ],
            "Resource": "*"
        }
    ]
}
```

Limiting a user to viewing specific canaries

You can limit a user's ability to view information about canaries, so that they can only see information about the canaries you specify. To do this, use an IAM policy with a `Condition` statement similar to the following, and attach this policy to a user or an IAM role.

The following example limits the user to only viewing information about `name-of-allowed-canary-1` and `name-of-allowed-canary-2`.

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "synthetics:DescribeCanaries",
            "Resource": "*",
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "synthetics:Names": [
                        "name-of-allowed-canary-1",
                        "name-of-allowed-canary-2"
                    ]
                }
            }
        }
    ]
}
```

CloudWatch Synthetics supports listing as many as five items in the `synthetics:Names` array.

You can also create a policy that uses a * as a wildcard in canary names that are to be allowed, as in the following example:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": "synthetics:DescribeCanaries",
            "Resource": "*",
            "Condition": {
                "ForAnyValue:StringLike": {
                    "synthetics:Names": [
                        "my-team-canary-*"
                    ]
                }
            }
        }
    ]
}
```

Any user signed in with one of these policies attached can't use the CloudWatch console to view any canary information. They can view canary information only for the canaries authorized by the policy and only by using the DescribeCanaries API or the describe-canaries Amazon CLI command.

Creating a canary

> **Important**
>
> Ensure that you use Synthetics canaries to monitor only endpoints and APIs where you have ownership or permissions. Depending on the canary frequency settings, these endpoints might experience increased traffic.

When you use the CloudWatch console to create a canary, you can use a blueprint provided by CloudWatch to create your canary or you can write your own script. For more information, see Using canary blueprints.

You can also create a canary using Amazon CloudFormation if you are using your own script for the canary. For more information, see AWS::Synthetics::Canary in the *Amazon CloudFormation User Guide*.

If you are writing your own script, you can use several functions that CloudWatch Synthetics has built into a library. For more information, see Synthetics runtime versions.

> **Note**
>
> When you create a canary, one of the layers created is a Synthetics layer prepended with `Synthetics`. This layer is owned by the Synthetics service account and contains the runtime code.

To create a canary

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Application Signals**, **Synthetics Canaries**.
3. Choose **Create Canary**.
4. Choose one of the following:

- To base your canary on a blueprint script, choose **Use a blueprint**, and then choose the type of canary you want to create. For more information about what each type of blueprint does, see Using canary blueprints.
- To upload your own Node.js script to create a custom canary, choose **Upload a script**.

  You can then drag your script into the **Script** area or choose **Browse files** to navigate to the script in your file system.
- To import your script from an S3 bucket, choose **Import from S3**. Under **Source location**, enter the complete path to your canary or choose **Browse S3**.

  You must have `s3:GetObject` and `s3:GetObjectVersion` permissions for the S3 bucket that you use. The bucket must be in the same Amazon Region where you are creating the canary.

5. Under **Name**, enter a name for your canary. The name is used on many pages, so we recommend that you give it a descriptive name that distinguishes it from other canaries.
6. Under **Application or endpoint URL**, enter the URL that you want the canary to test. This URL must include the protocol (such as https://).

   If you want the canary to test an endpoint on a VPC, you must also enter information about your VPC later in this procedure.
7. If you are using your own script for the canary, under **Lambda handler**, enter the entry point where you want the canary to start. If you use a runtime earlier than `syn-nodejs-puppeteer-3.4` or `syn-python-selenium-1.1`, the string that you enter must end with `.handler`. If you use `syn-nodejs-puppeteer-3.4` or `syn-python-selenium-1.1` or a later runtime, this restriction does not apply.
8. If you are using environment variables in your script, choose **Environment variables** and then specify a value for each environment variable defined in your script. For more information, see Environment variables.
9. Under **Schedule**, choose whether to run this canary just once, run it continuously using a rate expression, or schedule it using a cron expression.

   - When you use the CloudWatch console to create a canary that runs continuously, you can choose a rate anywhere between once a minute and once an hour.

- For more information about writing a cron expression for canary scheduling, see Scheduling canary runs using cron.

10. (Optional) To set a timeout value for the canary, choose **Additional configuration** and then specify the timeout value. Make it no shorter than 15 seconds to allow for Lambda cold starts and the time it takes to boot up the canary instrumentation.
11. Under **Data retention**, specify how long to retain information about both failed and successful canary runs. The range is 1-455 days.

    This setting affects the range of information returned by GetCanaryRuns operations, as well as the range of information displayed in the Synthetics console.

    It does not affect the data stored in your Amazon S3 buckets, or logs or metrics that are published by the canary.

    Regardless of the canary's data retention period, the range of information displayed in console has certain limits. In the Synthetics console home view, the relative and absolute time range are limited to seven days. In the Synthetics console view for a specific canary, the relative time range is limited to seven days and the absolute time range is limited to 30 days.
12. Under **Data Storage**, select the S3 bucket to use to store the data from the canary runs. The bucket name can't contain a period (.). If you leave this blank, a default S3 bucket is used or created.

    If you are using the `syn-nodejs-puppeteer-3.0` or later runtime, when you enter the URL for the bucket in the text box, you can specify a bucket in the current Region or in a different Region. If you are using an earlier runtime version, the bucket must be in the current Region.
13. (Optional) By default, canaries store their artifacts on Amazon S3, and the artifacts are encrypted at rest using an Amazon-managed Amazon KMS key. You can use a different encryption option by choosing **Additional configuration** in the **Data Storage** section. You can then choose the type of key to use for encryption. For more information, see Encrypting canary artifacts.
14. Under **Access permissions**, choose whether to create an IAM role to run the canary or use an existing one.

    If you have CloudWatch Synthetics create the role, it automatically includes all the necessary permissions. If you want to create the role yourself, see Required roles and permissions for canaries for information about the necessary permissions.

If you use the CloudWatch console to create a role for a canary when you create the canary, you can't re-use the role for other canaries, because these roles are specific to just one canary. If you have manually created a role that works for multiple canaries, you can use that existing role.

To use an existing role, you must have the `iam:PassRole` permission to pass that role to Synthetics and Lambda. You must also have the `iam:GetRole` permission.

15. (Optional) Under **Alarms**, choose whether you want default CloudWatch alarms to be created for this canary. If you choose to create alarms, they are created with the following name convention:`Synthetics-Alarm-canaryName-index`

    `index` is a number representing each different alarm created for this canary. The first alarm has an index of 1, the second alarm has an index of 2, and so on.
16. (Optional) To have this canary test an endpoint that is on a VPC, choose **VPC settings**, and then do the following:

    a. Select the VPC that hosts the endpoint.

    b. Select one or more subnets on your VPC. You must select a private subnet because a Lambda instance can't be configured to run in a public subnet when an IP address can't be assigned to the Lambda instance during execution. For more information, see Configuring a Lambda Function to Access Resources in a VPC.

    c. Select one or more security groups on your VPC.

    d. To allow outbound IPv6 traffic for this canary, select **Allow IPv6 Traffic for dual-stack subnets**. This enables the canary to monitor IPv6-only and dual stack enabled endpoints over IPv6.

       You can monitor endpoints external to your VPC by giving the canary internet access and configuring the VPC subnets appropriately. For more information, see Running a canary on a VPC.

    If the endpoint is on a VPC, you must enable your canary to send information to CloudWatch and Amazon S3. For more information, see Running a canary on a VPC.
17. (Optional) Under **Tags**, add one or more key-value pairs as tags for this canary. Tags can help you identify and organize your Amazon resources and track your Amazon costs. For more information, see Tagging your Amazon CloudWatch resources.

If you want the tags that you apply to the canary to also be applied to the Lambda function that the canary uses, choose **Lambda function** under **Tag Replication**. If you choose this option, CloudWatch Synthetics will keep the tags on the canary and the Lambda function synchronized:

- Synthetics will apply the same tags that you specify here to both your canary and your Lambda function.
- If you later update the canary's tags and keep this option selected, Synthetics modifies the tags on your Lambda function to remain in sync with the canary.

18. (Optional) Under **Active tracing**, choose whether to enable active X-Ray tracing for this canary. This option is available only if the canary uses runtime version **syn-nodejs-2.0** or later. **Active tracing** is only available for Puppeteer runtimes. For more information, see [Canaries and X-Ray tracing](#).

Resources that are created for canaries

When you create a canary, the following resources are created:

- An IAM role with the name `CloudWatchSyntheticsRole-canary-name-uuid` (if you use CloudWatch console to create the canary and specify for a new role to be created for the canary)
- An IAM policy with the name `CloudWatchSyntheticsPolicy-canary-name-uuid`.
- An S3 bucket with the name `cw-syn-results-accountID-region`.
- Alarms with the name `Synthetics-Alarm-MyCanaryName`, if you want alarms to be created for the canary.
- Lambda functions and layers, if you use a blueprint to create the canary. These resources have the prefix `cwsyn-MyCanaryName`.
- CloudWatch Logs log groups with the name `/aws/lambda/cwsyn-MyCanaryName-randomId`.

Using canary blueprints

This section provides details about each of the canary blueprints and the tasks each blueprint is best suited for. Blueprints are provided for the following canary types:

- Heartbeat Monitor
- API Canary

- Broken Link Checker
- Visual Monitoring
- Canary Recorder
- GUI Workflow

When you use a blueprint to create a canary, as you fill out the fields in the CloudWatch console, the **Script editor** area of the page displays the canary you are creating as a Node.js script. You can also edit your canary in this area to customize it further.

Heartbeat monitoring

Heartbeat scripts load the specified URL and store a screenshot of the page and an HTTP archive file (HAR file). They also store logs of accessed URLs.

You can use the HAR files to view detailed performance data about the web pages. You can analyze the list of web requests and catch performance issues such as time to load for an item.

If your canary uses the `syn-nodejs-puppeteer-3.1` or later runtime version, you can use the heartbeat monitoring blueprint to monitor multiple URLs and see the status, duration, associated screenshots, and failure reason for each URL in the step summary of the canary run report.

API canary

API canaries can test the basic Read and Write functions of a REST API. REST stands for *representational state transfer* and is a set of rules that developers follow when creating an API. One of these rules states that a link to a specific URL should return a piece of data.

Canaries can work with any APIs and test all types of functionality. Each canary can make multiple API calls.

In canaries that use runtime version `syn-nodejs-2.2` or later, the API canary blueprint supports multi-step canaries that monitor your APIs as HTTP steps. You can test multiple APIs in a single canary. Each step is a separate request that can access a different URL, use different headers, and use different rules for whether headers and response bodies are captured. By not capturing headers and response body, you can prevent sensitive data from being recorded.

Each request in an API canary consists of the following information:

- The *endpoint*, which is the URL that you request.

- The *method*, which is the type of request that is sent to the server. REST APIs support GET (read), POST (write), PUT (update), PATCH (update), and DELETE (delete) operations.
- The *headers*, which provide information to both the client and the server. They are used for authentication and providing information about the body content. For a list of valid headers, see [HTTP Headers](#).
- The *data* (or *body*), which contains information to be sent to the server. This is used only for POST, PUT, PATCH, or DELETE requests.

> **Note**
>
> API canary blueprints are not supported by Playwright runtimes.

The API canary blueprint supports GET and POST methods. When you use this blueprint, you must specify headers. For example, you can specify **`Authorization`** as a **Key** and specify the necessary authorization data as the **Value** for that key.

If you are testing a POST request, you also specify the content to post in the **Data** field.

Integration with API Gateway

The API blueprint is integrated with Amazon API Gateway. This enables you to select an API Gateway API and stage from the same Amazon account and Region as the canary, or to upload a Swagger template from API Gateway for cross-account and cross-Region API monitoring. You can then choose the rest of the details in the console to create the canary, instead of entering them from scratch. For more information about API Gateway, see [What is Amazon API Gateway?](#)

Using a private API

You can create a canary that uses a private API in Amazon API Gateway. For more information, see [Creating a private API in Amazon API Gateway?](#)

Broken link checker

The broken link checker collects all the links inside the URL that you are testing by using `document.getElementsByTagName('a')`. It tests only up to the number of links that you specify, and the URL itself is counted as the first link. For example, if you want to check all the links on a page that contains five links, you must specify for the canary to follow six links.

Broken link checker canaries created using the `syn-nodejs-2.0-beta` runtime or later support the following additional features:

- Provides a report that includes the links that were checked, status code, failure reason (if any), and source and destination page screenshots.
- When viewing canary results, you can filter to see only the broken links and then fix the link based on the reason for failure.
- This version captures annotated source page screenshots for each link and highlights the anchor where the link was found. Hidden components are not annotated.
- You can configure this version to capture screenshots of both source and destination pages, just source pages, or just destination pages.
- This version fixes an issue in the earlier version where the canary script stops after the first broken link even when more links are scraped from the first page.

> **Note**
>
> Broken link checker blueprints are not supported by Playwright runtimes.

To update an existing canary using `syn-1.0` to use the new runtime, you must delete and re-create the canary. Updating an existing canary to the new runtime does not make these features available.

A broken link checker canary detects the following types of link errors:

- 404 Page Not Found
- Invalid Host Name
- Bad URL. For example, the URL is missing a bracket, has extra slashes, or uses the wrong protocol.
- Invalid HTTP response code.
- The host server returns empty responses with no content and no response code.
- The HTTP requests constantly time out during the canary's run.
- The host consistently drops connections because it is misconfigured or is too busy.

Visual monitoring blueprint

The visual monitoring blueprint includes code to compare screenshots taken during a canary run with screenshots taken during a baseline canary run. If the discrepancy between the two screenshots is beyond a threshold percentage, the canary fails. Visual monitoring is supported in canaries running **syn-puppeteer-node-3.2** and later. It is not currently supported in canaries running Python and Selenium, or using Playwright runtimes.

The visual monitoring blueprint includes the following line of code in the default blueprint canary script, which enables visual monitoring.

```
syntheticsConfiguration.withVisualCompareWithBaseRun(true);
```

The first time that the canary runs successfully after this line is added to the script, it uses the screenshots taken during that run as the baseline for comparison. After that first canary run, you can use the CloudWatch console to edit the canary to do any of the following:

- Set the next run of the canary as the new baseline.
- Draw boundaries on the current baseline screenshot to designate areas of the screenshot to ignore during visual comparisons.
- Remove a screenshot from being used for visual monitoring.

For more information about using the CloudWatch console to edit a canary, see Edit or delete a canary.

You can also change the canary run that is used as the baseline by using the `nextrun` or `lastrun` parameters or specifing a canary run ID in the UpdateCanary API.

When you use the visual monitoring blueprint, you enter the URL where you want the screenshot to be taken, and specify a difference threshold as a percentage. After the baseline run, future runs of the canary that detect a visual difference greater than that threshold trigger a canary failure. After the baseline run, you can also edit the canary to "draw" boundaries on the baseline screenshot that you want to ignore during the visual monitoring.

The visual monitoring feature is powered by the the ImageMagick open source software toolkit. For more information, see ImageMagick .

Canary recorder

With the canary recorder blueprint, you can use the CloudWatch Synthetics Recorder to record your click and type actions on a website and automatically generate a Node.js script that can be used to create a canary that follows the same steps. The CloudWatch Synthetics Recorder is a Google Chrome extension provided by Amazon. The canary recorder is not supported for canaries that use the Playwright runtime.

Credits: The CloudWatch Synthetics Recorder is based on the [Headless recorder](#).

For more information, see [Using the CloudWatch Synthetics Recorder for Google Chrome](#).

GUI workflow builder

The GUI Workflow Builder blueprint verifies that actions can be taken on your webpage. For example, if you have a webpage with a login form, the canary can populate the user and password fields and submit the form to verify that the webpage is working correctly.

When you use a blueprint to create this type of canary, you specify the actions that you want the canary to take on the webpage. The actions that you can use are the following:

- **Click**— Selects the element that you specify and simulates a user clicking or choosing the element.

  To specify the element in a Node.js script, use `[id=]` or `a[class=]`.

  To specify the element in a Python script, use `xpath //*[@id=]` or `//*[@class=]`.
- **Verify selector**— Verifies that the specified element exists on the webpage. This test is useful for verifying that a previous action has causes the correct elements to populate the page.

  To specify the element to verify in a Node.js script, use `[id=]` or `a[class=]`.

  To specify the element to verify in a Python script, use `xpath //*[@id=]` or `//*[class=]`.
- **Verify text**— Verifies that the specified string is contained within the target element. This test is useful for verifying that a previous action has caused the correct text to be displayed.

  To specify the element in a Node.js script, use a format such as `div[@id=]//h1` because this action uses the `waitForXPath` function in Puppeteer.

  To specify the element in a Python script, use xpath format such as `//*[@id=]` or //*[@class=] because this action uses the `implicitly_wait` function in Selenium.

- **Input text**— Writes the specified text in the target element.

  To specify the element to verify in a Node.js script, use `[id=]` or `a[class=]`.

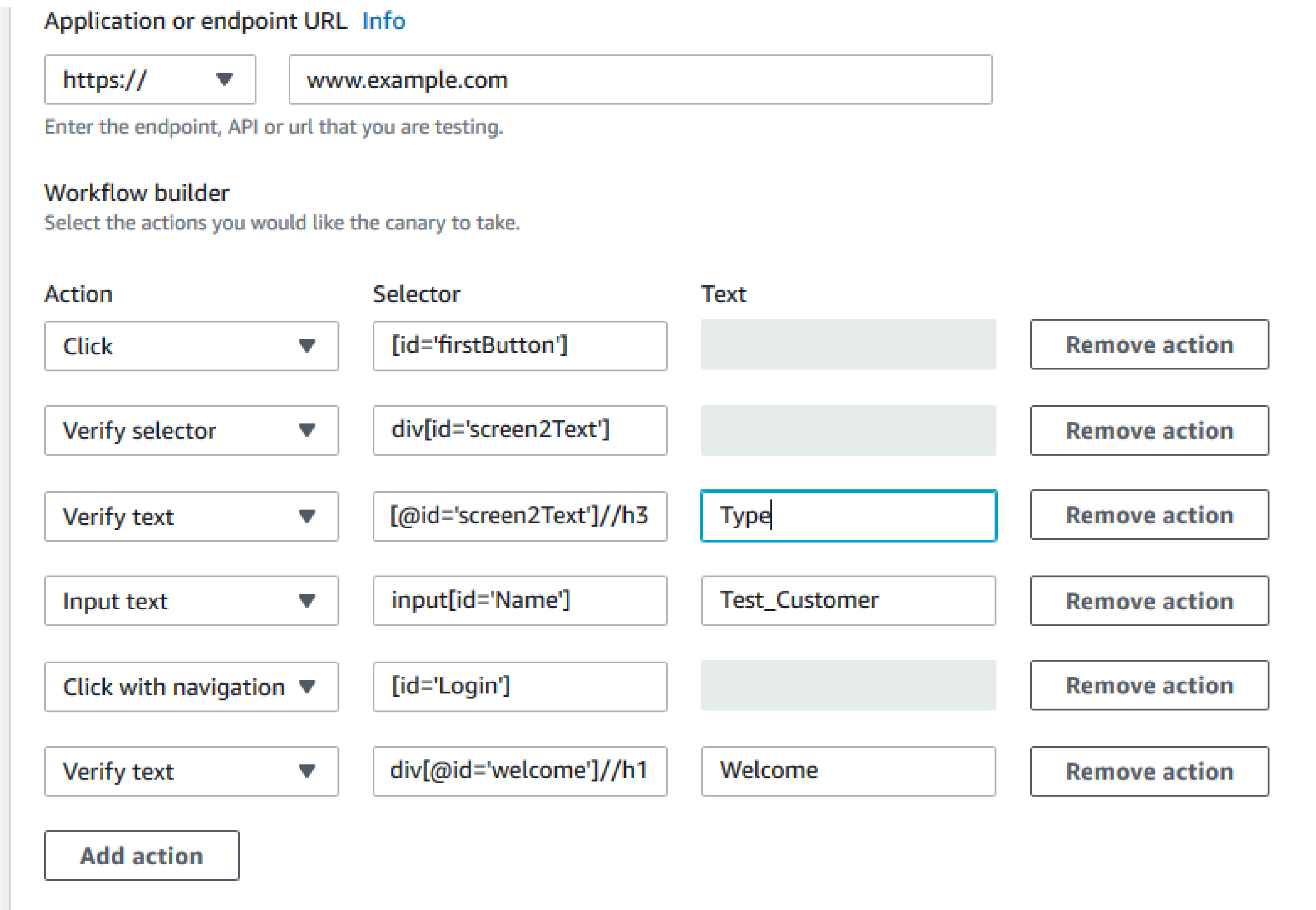
  To specify the element to verify in a Python script, use `xpath //*[@id=]` or `//*[@class=]`.

- **Click with navigation**— Waits for the whole page to load after choosing the specified element. This is most useful when you need to reload the page.

  To specify the element in a Node.js script, use `[id=]` or `a[class=]`.

  To specify the element in a Python script, use `xpath //*[@id=]` or `//*[@class=]`.

For example, the following blueprint uses Node.js. It clicks the **firstButton** on the specified URL, verifies that the expected selector with the expected text appears, inputs the name `Test_Customer` into the **Name** field, clicks the **Login** button, and then verifies that the login is successful by checking for the **Welcome** text on the next page.



GUI workflow canaries that use the following runtimes also provide a summary of the steps executed for each canary run. You can use the screenshots and error message associated with each step to find the root cause of failure.

- `syn-nodejs-2.0` or later
- `syn-python-selenium-1.0` or later

Using the CloudWatch Synthetics Recorder for Google Chrome

Amazon provides a CloudWatch Synthetics Recorder to help you create canaries more easily. The recorder is a Google Chrome extension.

The recorder records your click and type actions on a website and automatically generates a Node.js script that can be used to create a canary that follows the same steps.

After you start recording, the CloudWatch Synthetics Recorder detects your actions in the browser and converts them to a script. You can pause and resume the recording as needed. When you stop recording, the recorder produces a Node.js script of your actions, which you can easily copy with the **Copy to Clipboard** button. You can then use this script to create a canary in CloudWatch Synthetics.

Credits: The CloudWatch Synthetics Recorder is based on the Headless recorder.

Installing the CloudWatch Synthetics Recorder extension for Google Chrome

To use the CloudWatch Synthetics Recorder, you can start creating a canary and choose the **Canary Recorder** blueprint. If you do this when you haven't already downloaded the recorder, the CloudWatch Synthetics console provides a link to download it.

Alternatively, you can follow these steps to download and install the recorder directly.

To install the CloudWatch Synthetics Recorder

1. Using Google Chrome, go to this website: https://chrome.google.com/webstore/detail/cloudwatch-synthetics-rec/bhdnlmmgiplmbcdmkkdfplenecpegfno
2. Choose **Add to Chrome**, then choose **Add extension**.

Using the CloudWatch Synthetics Recorder for Google Chrome

To use the CloudWatch Synthetics Recorder to help you create a canary, you can choose **Create canary** in the CloudWatch console, and then choose **Use a blueprint**, **Canary Recorder**. For more information, see Creating a canary.

Alternatively, you can use the recorder to record steps without immediately using them to create a canary.

To use the CloudWatch Synthetics Recorder to record your actions on a website

1. Navigate to the page that you want to monitor.
2. Choose the Chrome extensions icon, and then choose **CloudWatch Synthetics Recorder**.
3. Choose **Start Recording**.
4. Perform the steps that you want to record. To pause recording, choose **Pause**.
5. When you are finished recording the workflow, choose **Stop recording**.
6. Choose **Copy to clipboard** to copy the generated script to your clipboard. Or, if you want to start over, choose **New recording**.
7. To create a canary with the copied script, you can paste your copied script into the recorder blueprint inline editor, or save it to an Amazon S3 bucket and import it from there.
8. If you're not immediately creating a canary, you can save your recorded script to a file.

Known limitations of the CloudWatch Synthetics Recorder

The CloudWatch Synthetics Recorder for Google Chrome currently has the following limitations.

- HTML elements that don’t have IDs will use CSS selectors. This can break canaries if the webpage structure changes later. We plan to provide some configuration options (such as using data-id) around this in a future version of the recorder.
- The recorder doesn't support actions such as double-click or copy/paste, and doesn't support key combinations such as CMD+0.
- To verify the presence of an element or text on the page, users must add assertions after the script is generated. The recorder doesn't support verifying an element without performing any action on that element. This is similar to the “Verify text” or “Verify element” options in the canary workflow builder. We plan to add some assertions support in a future version of the recorder.

- The recorder records all actions in the tab where the recording is initiated. It doesn't record pop-ups (for instance, to allow location tracking) or navigation to different pages from pop-ups.

Synthetics runtime versions

When you create or update a canary, you choose a Synthetics runtime version for the canary. A Synthetics runtime is a combination of the Synthetics code that calls your script handler, and the Lambda layers of bundled dependencies.

CloudWatch Synthetics currently supports runtimes that use Node.js for scripts and the Puppeteer framework, and runtimes that use Python for scripting and Selenium Webdriver for the framework.

We recommend that you always use the most recent runtime version for your canaries, to be able to use the latest features and updates made to the Synthetics library.

When you create a canary, one of the layers created is a Synthetics layer prefixed with `Synthetics`. This layer is owned by the Synthetics service account and contains the runtime code.

> **Note**
>
> Whenever you a canary to use a new version of the Synthetics runtime, all Synthetics library functions that your canary uses are also automatically u to the same version of NodeJS that the Synthetics runtime supports.

Topics

- CloudWatch Synthetics runtime support policy
- Runtime versions using Node.js and Playwright
- Runtime versions using Node.js and Puppeteer
- Runtime versions using Python and Selenium Webdriver

CloudWatch Synthetics runtime support policy

Synthetics runtime versions are subject to maintenance and security updates. When any component of a runtime version is no longer supported, that Synthetics runtime version is deprecated.

You can't create canaries using deprecated runtime versions. Canaries that use deprecated runtimes continue to run. You can stop, start, and delete these canaries. You can update an existing canary that uses a deprecated runtime version by updating the canary to use a supported runtime version.

CloudWatch Synthetics notifies you by email if you have canaries that use runtimes that are scheduled to be deprecated in the next 60 days. We recommend that you migrate your canaries to a supported runtime version to benefit from the new functionality, security, and performance enhancements that are included in more recent releases.

How do I update a canary to a new runtime version?

You can update a canary's runtime version by using the CloudWatch console, Amazon CloudFormation, the Amazon CLI or the Amazon SDK. When you use the CloudWatch console, you can update up to five canaries at once by selecting them in the canary list page and then choosing **Actions**, **Update Runtime**.

You can verify the update by first testing your update before committing the runtime update. When updating the runtime versions, choose the **Start Dry Run** or **Validate and save later** options in the CloudWatch console to create a dry run of the original canary along with any changes you made to the configuration. The dry run will update and execute the canary to validate whether the runtime update is safe for the canary. Once you have verified your canary with the new runtime version, you can update the runtime version of your canary. For more information, see Performing safe canary updates.

Alternatively, you can verify the update by first cloning the canary using the CloudWatch console and updating the runtime version. This creates another canary which is a clone of your original canary. Once you have verified your canary with the new runtime version, you can update the runtime version of your original canary and delete the clone canary.

You can also update multiple canaries using an upgrade script. For more information, see Canary runtime upgrade script.

If you upgrade a canary and it fails, see Troubleshooting a failed canary.

CloudWatch Synthetics runtime deprecation dates

The following table lists the date of deprecation of each deprecated CloudWatch Synthetics runtime.

Runtime Version	Deprecation date
`syn-nodejs-puppeteer-7.0`	October 1, 2025
`syn-nodejs-puppeteer-6.2`	October 1, 2025
`syn-nodejs-puppeteer-5.2`	October 1, 2025
`syn-python-selenium-3.0`	October 1, 2025
`syn-python-selenium-2.1`	October 1, 2025
`syn-nodejs-puppeteer-6.1`	March 8, 2024
`syn-nodejs-puppeteer-6.0`	March 8, 2024
`syn-nodejs-puppeteer-5.1`	March 8, 2024
`syn-nodejs-puppeteer-5.0`	March 8, 2024
`syn-nodejs-puppeteer-4.0`	March 8, 2024
`syn-nodejs-puppeteer-3.9`	January 8, 2024
`syn-nodejs-puppeteer-3.8`	January 8, 2024

Runtime Version	Deprecation date
syn-python-selenium-2.0	March 8, 2024
syn-python-selenium-1.3	March 8, 2024
syn-python-selenium-1.2	March 8, 2024
syn-python-selenium-1.1	March 8, 2024
syn-python-selenium-1.0	March 8, 2024
syn-nodejs-puppeteer-3.7	January 8, 2024
syn-nodejs-puppeteer-3.6	January 8, 2024
syn-nodejs-puppeteer-3.5	January 8, 2024
syn-nodejs-puppeteer-3.4	November 13, 2022
syn-nodejs-puppeteer-3.3	November 13, 2022
syn-nodejs-puppeteer-3.2	November 13, 2022
syn-nodejs-puppeteer-3.1	November 13, 2022

Runtime Version	Deprecation date
syn-nodejs-puppeteer-3.0	November 13, 2022
syn-nodejs-2.2	May 28, 2021
syn-nodejs-2.1	May 28, 2021
syn-nodejs-2.0	May 28, 2021
syn-nodejs-2.0-beta	February 8, 2021
syn-1.0	May 28, 2021

Canary runtime upgrade script

To upgrade a canary script to a supported runtime version, use the following script.

```
const AWS = require('aws-sdk');

// You need to configure your Amazon credentials and Region.
//   https://docs.aws.amazon.com/sdk-for-javascript/v3/developer-guide/setting-credentials-node.html
//   https://docs.aws.amazon.com/sdk-for-javascript/v3/developer-guide/setting-region.html

const synthetics = new AWS.Synthetics();

const DEFAULT_OPTIONS = {
  /**
   * The number of canaries to upgrade during a single run of this script.
   */
  count: 10,
  /**
   * No canaries are upgraded unless force is specified.
   */
  force: false
};
```

```
/**
 * The number of milliseconds to sleep between GetCanary calls when
 * verifying that an update succeeded.
 */
const SLEEP_TIME = 5000;

(async () => {
  try {
    const options = getOptions();

    const versions = await getRuntimeVersions();
    const canaries = await getAllCanaries();
    const upgrades = canaries
      .filter(canary => !versions.isLatestVersion(canary.RuntimeVersion))
      .map(canary => {
        return {
          Name: canary.Name,
          FromVersion: canary.RuntimeVersion,
          ToVersion: versions.getLatestVersion(canary.RuntimeVersion)
        };
      });

    if (options.force) {
      const promises = [];

      for (const upgrade of upgrades.slice(0, options.count)) {
        const promise = upgradeCanary(upgrade);
        promises.push(promise);
        // Sleep for 100 milliseconds to avoid throttling.
        await usleep(100);
      }

      const succeeded = [];
      const failed = [];
      for (let i = 0; i < upgrades.slice(0, options.count).length; i++) {
        const upgrade = upgrades[i];
        const promise = promises[i];
        try {
          await promise;
          console.log(`The update of ${upgrade.Name} succeeded.`);
          succeeded.push(upgrade.Name);
        } catch (e) {
          console.log(`The update of ${upgrade.Name} failed with error: ${e}`);
          failed.push({
```

```
            Name: upgrade.Name,
            Reason: e
          });
        }
      }

      if (succeeded.length) {
        console.group('The following canaries were upgraded successfully.');
        for (const name of succeeded) {
          console.log(name);
        }
        console.groupEnd()
      } else {
        console.log('No canaries were upgraded successfully.');
      }

      if (failed.length) {
        console.group('The following canaries were not upgraded successfully.');
        for (const failure of failed) {
          console.log('\x1b[31m', `${failure.Name}: ${failure.Reason}`, '\x1b[0m');
        }
        console.groupEnd();
      }
    } else {
      console.log('Run with --force [--count <count>] to perform the first <count>
 upgrades shown. The default value of <count> is 10.')
      console.table(upgrades);
    }
  } catch (e) {
    console.error(e);
  }
})();

function getOptions() {
  const force = getFlag('--force', DEFAULT_OPTIONS.force);
  const count = getOption('--count', DEFAULT_OPTIONS.count);
  return { force, count };

  function getFlag(key, defaultValue) {
    return process.argv.includes(key) || defaultValue;
  }
  function getOption(key, defaultValue) {
    const index = process.argv.indexOf(key);
    if (index < 0) {
```

```
      return defaultValue;
    }
    const value = process.argv[index + 1];
    if (typeof value === 'undefined' || value.startsWith('-')) {
      throw `The ${key} option requires a value.`;
    }
    return value;
  }
}

function getAllCanaries() {
  return new Promise((resolve, reject) => {
    const canaries = [];

    synthetics.describeCanaries().eachPage((err, data) => {
      if (err) {
        reject(err);
      } else {
        if (data === null) {
          resolve(canaries);
        } else {
          canaries.push(...data.Canaries);
        }
      }
    });
  });
}

function getRuntimeVersions() {
  return new Promise((resolve, reject) => {
    const jsVersions = [];
    const pythonVersions = [];
    synthetics.describeRuntimeVersions().eachPage((err, data) => {
      if (err) {
        reject(err);
      } else {
        if (data === null) {
          jsVersions.sort((a, b) => a.ReleaseDate - b.ReleaseDate);
          pythonVersions.sort((a, b) => a.ReleaseDate - b.ReleaseDate);
          resolve({
            isLatestVersion(version) {
              const latest = this.getLatestVersion(version);
              return latest === version;
            },
```

```
            getLatestVersion(version) {
              if (jsVersions.some(v => v.VersionName === version)) {
                return jsVersions[jsVersions.length - 1].VersionName;
              } else if (pythonVersions.some(v => v.VersionName === version)) {
                return pythonVersions[pythonVersions.length - 1].VersionName;
              } else {
                throw Error(`Unknown version ${version}`);
              }
            }
          });
        } else {
          for (const version of data.RuntimeVersions) {
            if (version.VersionName === 'syn-1.0') {
              jsVersions.push(version);
            } else if (version.VersionName.startsWith('syn-nodejs-2.')) {
              jsVersions.push(version);
            } else if (version.VersionName.startsWith('syn-nodejs-puppeteer-')) {
              jsVersions.push(version);
            } else if (version.VersionName.startsWith('syn-python-selenium-')) {
              pythonVersions.push(version);
            } else {
              throw Error(`Unknown version ${version.VersionName}`);
            }
          }
        }
      }
    });
  });
}

async function upgradeCanary(upgrade) {
  console.log(`Upgrading canary ${upgrade.Name} from ${upgrade.FromVersion} to
 ${upgrade.ToVersion}`);
  await synthetics.updateCanary({ Name: upgrade.Name, RuntimeVersion:
 upgrade.ToVersion }).promise();
  while (true) {
    await usleep(SLEEP_TIME);
    console.log(`Getting the state of canary ${upgrade.Name}`);
    const response = await synthetics.getCanary({ Name: upgrade.Name }).promise();
    const state = response.Canary.Status.State;
    console.log(`The state of canary ${upgrade.Name} is ${state}`);
    if (state === 'ERROR' || response.Canary.Status.StateReason) {
      throw response.Canary.Status.StateReason;
    }
```

```
    if (state !== 'UPDATING') {
      return;
    }
  }
}

function usleep(ms) {
  return new Promise(resolve => setTimeout(resolve, ms));
}
```

Runtime versions using Node.js and Playwright

The following sections contain information about the CloudWatch Synthetics runtime versions for Node.js and Playwright. Playwright is an open-source automation library for browser testing. For more information about Playwright, see https://playwright.dev/

The naming convention for these runtime versions is `syn-*language*-*framework*-*majorversion*.*minorversion*`.

syn-nodejs-playwright-2.0

Major dependencies:

- Amazon Lambda runtime Node.js 20.x
- Playwright version 1.49.1
- Playwright/test version 1.49.1
- Chromium version 131.0.6778.264

Changes in syn-nodejs-playwright-2.0

- The mismatch between total duration and sum of timings for a given request in HAR file is fixed.
- Supports dry runs for the canary which allows for adhoc executions or performing a safe canary update.

Previous runtime versions for Node.js and Playwright

The following earlier runtime versions for Node.js and Playwright are still supported.

syn-nodejs-playwright-1.0

Major dependencies:

- Amazon Lambda runtime Node.js 20.x
- Playwright version 1.44.1
- Playwright/test version 1.44.1
- Chromium version 126.0.6478.126

Features:

- **PlayWright support –** You can write canary scripts by using the Playwright automation framework. You can bring your existing Playwright scripts to run as canaries, and enhance them with Amazon monitoring capabilities.
- **CloudWatch Logs integration –** You can query and filter for logs through the CloudWatch Synthetics console. Each log message contains unique `canaryRunId`, making it easy to search for logs for a particular canary run.
- **Metrics and canary artifacts –** You can monitor canary run pass rate through CloudWatch metrics, and configure alarms to alert you when canaries detect issues.
- **Screenshots and steps association –** You can capture screenshots using native Playwright functionality to visualize the stages of a canary script on each run. Screenshots are automatically associated with canary steps, and are uploaded to Amazon S3 buckets.
- **Multiple tabs–** You can create canaries that open multiple browser tabs, and access screenshots from each tab. You can create multi-tab and multi-step user workflows in Synthetics.

Runtime versions using Node.js and Puppeteer

The first runtime version for Node.js and Puppeteer was named `syn-1.0`. Later runtime versions have the naming convention `syn-`*`language`*`-`*`majorversion`*`.`*`minorversion`*. Starting with `syn-nodejs-puppeteer-3.0`, the naming convention is `syn-`*`language`*`-`*`framework`*`-`*`majorversion`*`.`*`minorversion`*

An additional `-beta` suffix shows that the runtime version is currently in a beta preview release.

Runtime versions with the same major version number are backward compatible.

> **Important**
>
> IMPORTANT: The included Amazon SDK for JavaScript v2 dependency will be removed and updated to use Amazon SDK for JavaScript v3 in a future runtime release. When that happens, you can update your canary code references. Alternatively, you can continue referencing and using the included Amazon SDK for JavaScript v2 dependency by adding it as a dependency to your source code zip file.

The Lambda code in a canary is configured to have a maximum memory of 1 GB. Each run of a canary times out after a configured timeout value. If no timeout value is specified for a canary, CloudWatch chooses a timeout value based on the canary's frequency. If you configure a timeout value, make it no shorter than 15 seconds to allow for Lambda cold starts and the time it takes to boot up the canary instrumentation.

Notes for all runtime versions

When using `syn-nodejs-puppeteer-3.0` runtime version, make sure that your canary script is compatible with Node.js 12.x. If you use an earlier version of a `syn-nodejs` runtime version, make sure that that your script is compatible with Node.js 10.x.

syn-nodejs-puppeteer-10.0

`syn-nodejs-puppeteer-10.0` is the most recent Synthetics runtime for Node.js and Puppeteer.

> **Important**
>
> Lambda Node.js 18 and later runtimes use Amazon SDK for JavaScript V3. If you need to migrate a function from an earlier runtime, follow the aws-sdk-js-v3 Migration Workshop on GitHub. For more information about Amazon SDK for JavaScript version 3, see this blog post.

Major dependencies:

- Lambda runtime Node.js 20.x
- Puppeteer-core version 24.2.0
- Chromium version 131.0.6778.264

Changes in syn-nodejs-puppeteer-10.0

- The bug related to closing the browser that took excessively long is fixed.
- Supports dry runs for the canary which allows for adhoc executions or performing a safe canary update.

Previous runtime versions for Node.js and Puppeteer

The following earlier runtime versions for Node.js and Puppeteer are still supported.

syn-nodejs-puppeteer-9.1

`syn-nodejs-puppeteer-9.1` is the most recent Synthetics runtime for Node.js and Puppeteer.

> **Important**
>
> Lambda Node.js 18 and later runtimes use Amazon SDK for JavaScript V3. If you need to migrate a function from an earlier runtime, follow the aws-sdk-js-v3 Migration Workshop on GitHub. For more information about Amazon SDK for JavaScript version 3, see this blog post.

Major dependencies:

- Lambda runtime Node.js 20.x
- Puppeteer-core version 22.12.1
- Chromium version 126.0.6478.126

Changes in syn-nodejs-puppeteer-9.1 – Bug fixes related to date ranges and pending requests in HAR files are fixed.

syn-nodejs-puppeteer-9.0

> **Important**
>
> Lambda Node.js 18 and later runtimes use Amazon SDK for JavaScript V3. If you need to migrate a function from an earlier runtime, follow the aws-sdk-js-v3 Migration Workshop

on GitHub. For more information about Amazon SDK for JavaScript version 3, see this blog post.

Major dependencies:

- Lambda runtime Node.js 20.x
- Puppeteer-core version 22.12.1
- Chromium version 126.0.6478.126

Changes in syn-nodejs-puppeteer-9.0 – The bug fix to enable visual monitoring capabilities is fixed.

syn-nodejs-puppeteer-8.0

> **Warning**
>
> Because of a bug, the `syn-nodejs-puppeteer-8.0` runtime doesn't support visual monitoring in canaries. Upgrade to syn-nodejs-puppeteer-9.0 for the bug fix for visual monitoring.

> **Important**
>
> Lambda Node.js 18 and later runtimes use Amazon SDK for JavaScript V3. If you need to migrate a function from an earlier runtime, follow the aws-sdk-js-v3 Migration Workshop on GitHub. For more information about Amazon SDK for JavaScript version 3, see this blog post.

Major dependencies:

- Lambda runtime Node.js 20.x
- Puppeteer-core version 22.10.0
- Chromium version 125.0.6422.112

Updates in syn-nodejs-puppeteer-8.0:

- **Support for two-factor authentication**
- **Bug fixes** related to some service clients losing data in Node.js SDK V3 responses is fixed.

syn-nodejs-puppeteer-7.0

Major dependencies:

- Lambda runtime Node.js 18.x
- Puppeteer-core version 21.9.0
- Chromium version 121.0.6167.139

Code size:

The size of code and dependencies that you can package into this runtime is 80 MB.

Updates in syn-nodejs-puppeteer-7.0:

- **Updated versions of the bundled libraries in Puppeteer and Chromium**— The Puppeteer and Chromium dependencies are updated to new versions.

> **Important**
>
> Moving from Puppeteer 19.7.0 to Puppeteer 21.9.0 introduces breaking changes regarding testing and filters. For more information, see the **BREAKING CHANGES** sections in puppeteer: v20.0.0 and puppeteer-core: v21.0.0.

Recommended upgrade to Amazon SDK v3

The Lambda nodejs18.x runtime doesn't support Amazon SDK v2. We strongly recommend that you migrate to Amazon SDK v3.

syn-nodejs-puppeteer-6.2

Major dependencies:

- Lambda runtime Node.js 18.x
- Puppeteer-core version 19.7.0

- Chromium version 111.0.5563.146

Changes in syn-nodejs-puppeteer-6.2:

- **Updated versions of the bundled libraries in Chromium**
- **Ephemeral storage monitoring**— This runtime adds ephemeral storage monitoring in customer accounts.
- **Bug fixes**

syn-nodejs-puppeteer-5.2

Major dependencies:

- Lambda runtime Node.js 16.x
- Puppeteer-core version 19.7.0
- Chromium version 111.0.5563.146

Updates in syn-nodejs-puppeteer-5.2:

- **Updated versions of the bundled libraries in Chromium**
- **Bug fixes**

Deprecated runtime versions for Node.js and Puppeteer

The following runtimes for Node.js and Puppeteer have been deprecated. For information about runtime deprecation dates, see CloudWatch Synthetics runtime deprecation dates.

syn-nodejs-puppeteer-6.1

Major dependencies:

- Lambda runtime Node.js 18.x
- Puppeteer-core version 19.7.0
- Chromium version 111.0.5563.146

Updates in syn-nodejs-puppeteer-6.1:

- **Stability improvements**— Added auto-retry logic for handling intermittent Puppeteer launch errors.
- **Dependency upgrades**— Upgrades for some third-party dependency packages.
- **Canaries without Amazon S3 permissions**— Bug fixes, such that canaries that don't have any Amazon S3 permissions can still run. These canaries with no Amazon S3 permissions won't be able to upload screenshots or other artifacts to Amazon S3. For more information about permissions for canaries, see [Required roles and permissions for canaries](#).

> **Important**
>
> IMPORTANT: The included Amazon SDK for JavaScript v2 dependency will be removed and updated to use Amazon SDK for JavaScript v3 in a future runtime release. When that happens, you can update your canary code references. Alternatively, you can continue referencing and using the included Amazon SDK for JavaScript v2 dependency by adding it as a dependency to your source code zip file.

syn-nodejs-puppeteer-6.0

Major dependencies:

- Lambda runtime Node.js 18.x
- Puppeteer-core version 19.7.0
- Chromium version 111.0.5563.146

Updates in syn-nodejs-puppeteer-6.0:

- **Dependency upgrade**— The Node.js dependency is upgraded to 18.x.
- **Intercept mode support**— Puppeteer cooperative intercept mode support was added to the Synthetics canary runtime library.
- **Tracing behavior change**— Changed default tracing behavior to trace only fetch and xhr requests, and not trace resource requests. You can enable the tracing of resource requests by configuring the `traceResourceRequests` option.
- **Duration metric refined**— The `Duration` metric now excludes the operation time the canary uses to upload artifacts, take screenshots, and generate CloudWatch metrics. `Duration` metric values are reported to CloudWatch, and you can also see them in the Synthetics console.

- **Bug fix**— Clean up core dump generated when Chromium crashes during a canary run.

> **Important**
>
> IMPORTANT: The included Amazon SDK for JavaScript v2 dependency will be removed and updated to use Amazon SDK for JavaScript v3 in a future runtime release. When that happens, you can update your canary code references. Alternatively, you can continue referencing and using the included Amazon SDK for JavaScript v2 dependency by adding it as a dependency to your source code zip file.

syn-nodejs-puppeteer-5.1

Major dependencies:

- Lambda runtime Node.js 16.x
- Puppeteer-core version 19.7.0
- Chromium version 111.0.5563.146

Bug fixes in syn-nodejs-puppeteer-5.1:

- **Bug fix**— This runtime fixes a bug in `syn-nodejs-puppeteer-5.0` where the HAR files created by the canaries were missing request headers.

syn-nodejs-puppeteer-5.0

Major dependencies:

- Lambda runtime Node.js 16.x
- Puppeteer-core version 19.7.0
- Chromium version 111.0.5563.146

Updates in syn-nodejs-puppeteer-5.0:

- **Dependency upgrade**— The Puppeteer-core version is updated to 19.7.0. The Chromium version is upgraded to 111.0.5563.146.

> **Important**
>
> The new Puppeteer-core version is not completely backward-compatible with previous versions of Puppeteer. Some of the changes in this version can cause existing canaries that use deprecated Puppeteer functions to fail. For more information, see the breaking changes in the change logs for Puppeteer-core versions 19.7.0 through 6.0, in Puppeteer change logs.

syn-nodejs-puppeteer-4.0

Major dependencies:

- Lambda runtime Node.js 16.x
- Puppeteer-core version 5.5.0
- Chromium version 92.0.4512

Updates in syn-nodejs-puppeteer-4.0:

- **Dependency upgrade**— The Node.js dependency is updated to 16.x.

syn-nodejs-puppeteer-3.9

> **Important**
>
> This runtime version was deprecated on January 8, 2024. For more information, see CloudWatch Synthetics runtime support policy.

Major dependencies:

- Lambda runtime Node.js 14.x
- Puppeteer-core version 5.5.0
- Chromium version 92.0.4512

Updates in syn-nodejs-puppeteer-3.9:

- **Dependency upgrades**— Upgrades some third-party dependency packages.

syn-nodejs-puppeteer-3.8

> **Important**
>
> This runtime version was deprecated on January 8, 2024. For more information, see [CloudWatch Synthetics runtime support policy](#).

Major dependencies:

- Lambda runtime Node.js 14.x
- Puppeteer-core version 5.5.0
- Chromium version 92.0.4512

Updates in syn-nodejs-puppeteer-3.8:

- **Profile cleanup**— Chromium profiles are now cleaned up after each canary run.

Bug fixes in syn-nodejs-puppeteer-3.8:

- **Bug fixes**— Previously, visual monitoring canaries would sometimes stop working properly after a run with no screenshots. This is now fixed.

syn-nodejs-puppeteer-3.7

> **Important**
>
> This runtime version was deprecated on January 8, 2024. For more information, see [CloudWatch Synthetics runtime support policy](#).

Major dependencies:

- Lambda runtime Node.js 14.x
- Puppeteer-core version 5.5.0

- Chromium version 92.0.4512

Updates in syn-nodejs-puppeteer-3.7:

- **Logging enhancement**— The canary will upload logs to Amazon S3 even if it times out or crashes.
- **Lambda layer size reduced**— The size of the Lambda layer used for canaries is reduced by 34%.

Bug fixes in syn-nodejs-puppeteer-3.7:

- **Bug fixes**— Japanese, Simplified Chinese, and Traditional Chinese fonts will render properly.

syn-nodejs-puppeteer-3.6

> **Important**
>
> This runtime version was deprecated on January 8, 2024. For more information, see CloudWatch Synthetics runtime support policy.

Major dependencies:

- Lambda runtime Node.js 14.x
- Puppeteer-core version 5.5.0
- Chromium version 92.0.4512

Updates in syn-nodejs-puppeteer-3.6:

- **More precise timestamps**— The start time and stop time of canary runs are now precise to the millisecond.

syn-nodejs-puppeteer-3.5

> **Important**
>
> This runtime version was deprecated on January 8, 2024. For more information, see CloudWatch Synthetics runtime support policy.

Major dependencies:

- Lambda runtime Node.js 14.x
- Puppeteer-core version 5.5.0
- Chromium version 92.0.4512

Updates in syn-nodejs-puppeteer-3.5:

- **Updated dependencies**— The only new features in this runtime are the updated dependencies.

syn-nodejs-puppeteer-3.4

> **Important**
>
> This runtime version was deprecated on November 13, 2022. For more information, see CloudWatch Synthetics runtime support policy.

Major dependencies:

- Lambda runtime Node.js 12.x
- Puppeteer-core version 5.5.0
- Chromium version 88.0.4298.0

Updates in syn-nodejs-puppeteer-3.4:

- **Custom handler function**— You can now use a custom handler function for your canary scripts. Previous runtimes required the script entry point to include `.handler`.

You can also put canary scripts in any folder and pass the folder name as part of the handler. For example, `MyFolder/MyScriptFile.functionname` can be used as an entry point.

- **Expanded HAR file information**— You can now see bad, pending, and incomplete requests in the HAR files produced by canaries.

syn-nodejs-puppeteer-3.3

> **Important**
>
> This runtime version was deprecated on November 13, 2022. For more information, see [CloudWatch Synthetics runtime support policy](#).

Major dependencies:

- Lambda runtime Node.js 12.x
- Puppeteer-core version 5.5.0
- Chromium version 88.0.4298.0

Updates in syn-nodejs-puppeteer-3.3:

- **More options for artifact encryption**— For canaries using this runtime or later, instead of using an Amazon managed key to encrypt artifacts that the canary stores in Amazon S3, you can choose to use an Amazon KMS customer managed key or an Amazon S3-managed key. For more information, see [Encrypting canary artifacts](#).

syn-nodejs-puppeteer-3.2

> **Important**
>
> This runtime version was deprecated on November 13, 2022. For more information, see [CloudWatch Synthetics runtime support policy](#).

Major dependencies:

- Lambda runtime Node.js 12.x
- Puppeteer-core version 5.5.0
- Chromium version 88.0.4298.0

Updates in syn-nodejs-puppeteer-3.2:

- **visual monitoring with screenshots**— Canaries using this runtime or later can compare a screenshot taken during a run with a baseline version of the same screenshot. If the screenshots are more different than a specified percentage threshold, the canary fails. For more information, see Visual monitoring or Visual monitoring blueprint.
- **New functions regarding sensitive data** You can prevent sensitive data from appearing in canary logs and reports. For more information, see SyntheticsLogHelper class.
- **Deprecated function** The `RequestResponseLogHelper` class is deprecated in favor of other new configuration options. For more information, see RequestResponseLogHelper class.

syn-nodejs-puppeteer-3.1

> **Important**
>
> This runtime version was deprecated on November 13, 2022. For more information, see CloudWatch Synthetics runtime support policy.

Major dependencies:

- Lambda runtime Node.js 12.x
- Puppeteer-core version 5.5.0
- Chromium version 88.0.4298.0

Updates in syn-nodejs-puppeteer-3.1:

- **Ability to configure CloudWatch metrics**— With this runtime, you can disable the metrics that you do not require. Otherwise, canaries publish various CloudWatch metrics for each canary run.
- **Screenshot linking**— You can link a screenshot to a canary step after the step has completed. To do this, you take the screenshot by using the **takeScreenshot** method, using the name of the

step that you want to associate the screenshot with. For example, you might want to perform a step, add a wait time, and then take the screenshot.

- **Heartbeat monitor blueprint can monitor multiple URLs**— You can use the heartbeat monitoring blueprint in the CloudWatch console to monitor multiple URLs and see the status, duration, associated screenshots, and failure reason for each URL in the step summary of the canary run report.

syn-nodejs-puppeteer-3.0

> **Important**
>
> This runtime version was deprecated on November 13, 2022. For more information, see CloudWatch Synthetics runtime support policy.

Major dependencies:

- Lambda runtime Node.js 12.x
- Puppeteer-core version 5.5.0
- Chromium version 88.0.4298.0

Updates in syn-nodejs-puppeteer-3.0:

- **Upgraded dependencies**— This version uses Puppeteer version 5.5.0, Node.js 12.x, and Chromium 88.0.4298.0.
- **Cross-Region bucket access**— You can now specify an S3 bucket in another Region as the bucket where your canary stores its log files, screenshots, and HAR files.
- **New functions available**— This version adds library functions to retrieve the canary name and the Synthetics runtime version.

  For more information, see Synthetics class.

syn-nodejs-2.2

This section contains information about the `syn-nodejs-2.2` runtime version.

> **Important**
>
> This runtime version was deprecated on May 28, 2021. For more information, see CloudWatch Synthetics runtime support policy.

Major dependencies:

- Lambda runtime Node.js 10.x
- Puppeteer-core version 3.3.0
- Chromium version 83.0.4103.0

Changes in syn-nodejs-2.2:

- **Monitor your canaries as HTTP steps**— You can now test multiple APIs in a single canary. Each API is tested as a separate HTTP step, and CloudWatch Synthetics monitors the status of each step using step metrics and the CloudWatch Synthetics step report. CloudWatch Synthetics creates `SuccessPercent` and `Duration` metrics for each HTTP step.

  This functionality is implemented by the **executeHttpStep(stepName, requestOptions, callback, stepConfig)** function. For more information, see executeHttpStep(stepName, requestOptions, [callback], [stepConfig]).

  The API canary blueprint is updated to use this new feature.
- **HTTP request reporting**— You can now view detailed HTTP requests reports which capture details such as request/response headers, response body, status code, error and performance timings, TCP connection time, TLS handshake time, first byte time, and content transfer time. All HTTP requests which use the HTTP/HTTPS module under the hood are captured here. Headers and response body are not captured by default but can be enabled by setting configuration options.
- **Global and step-level configuration**— You can set CloudWatch Synthetics configurations at the global level, which are applied to all steps of canaries. You can also override these configurations at the step level by passing configuration key/value pairs to enable or disable certain options.

  For more information, see SyntheticsConfiguration class.

- **Continue on step failure configuration**— You can choose to continue canary execution when a step fails. For the `executeHttpStep` function, this is turned on by default. You can set this option once at global level or set it differently per-step.

syn-nodejs-2.1

> **Important**
>
> This runtime version was deprecated on May 28, 2021. For more information, see CloudWatch Synthetics runtime support policy.

Major dependencies:

- Lambda runtime Node.js 10.x
- Puppeteer-core version 3.3.0
- Chromium version 83.0.4103.0

Updates in syn-nodejs-2.1:

- **Configurable screenshot behavior**— Provides the ability to turn off the capturing of screenshots by UI canaries. In canaries that use previous versions of the runtimes, UI canaries always capture screenshots before and after each step. With `syn-nodejs-2.1`, this is configurable. Turning off screenshots can reduce your Amazon S3 storage costs, and can help you comply with HIPAA regulations. For more information, see SyntheticsConfiguration class.
- **Customize the Google Chrome launch parameters** You can now configure the arguments used when a canary launches a Google Chrome browser window. For more information, see launch(options).

There can be a small increase in canary duration when using syn-nodejs-2.0 or later, compared to earlier versions of the canary runtimes.

syn-nodejs-2.0

Important

This runtime version was deprecated on May 28, 2021. For more information, see CloudWatch Synthetics runtime support policy.

Major dependencies:

- Lambda runtime Node.js 10.x
- Puppeteer-core version 3.3.0
- Chromium version 83.0.4103.0

Updates in syn-nodejs-2.0:

- **Upgraded dependencies**— This runtime version uses Puppeteer-core version 3.3.0 and Chromium version 83.0.4103.0
- **Support for X-Ray active tracing.** When a canary has tracing enabled, X-Ray traces are sent for all calls made by the canary that use the browser, the Amazon SDK, or HTTP or HTTPS modules. Canaries with tracing enabled appear on the X-Ray Trace Map, even when they don't send requests to other services or applications that have tracing enabled. For more information, see Canaries and X-Ray tracing.
- **Synthetics reporting**— For each canary run, CloudWatch Synthetics creates a report named `SyntheticsReport-PASSED.json` or `SyntheticsReport-FAILED.json` which records data such as start time, end time, status, and failures. It also records the PASSED/FAILED status of each step of the canary script, and failures and screenshots captured for each step.
- **Broken link checker report**— The new version of the broken link checker included in this runtime creates a report that includes the links that were checked, status code, failure reason (if any), and source and destination page screenshots.
- **New CloudWatch metrics**— Synthetics publishes metrics named `2xx`, `4xx`, `5xx`, and `RequestFailed` in the `CloudWatchSynthetics` namespace. These metrics show the number of 200s, 400s, 500s, and request failures in the canary runs. With this runtime version, these metrics are reported only for UI canaries, and are not reported for API canaries. They are also reported for API canaries starting with runtime version `syn-nodejs-puppeteeer-2.2`.
- **Sortable HAR files**— You can now sort your HAR files by status code, request size, and duration.

- **Metrics timestamp**— CloudWatch metrics are now reported based on the Lambda invocation time instead of the canary run end time.

Bug fixes in syn-nodejs-2.0:

- Fixed the issue of canary artifact upload errors not being reported. Such errors are now surfaced as execution errors.
- Fixed the issue of redirected requests (3xx) being incorrectly logged as errors.
- Fixed the issue of screenshots being numbered starting from 0. They should now start with 1.
- Fixed the issue of screenshots being garbled for Chinese and Japanese fonts.

There can be a small increase in canary duration when using syn-nodejs-2.0 or later, compared to earlier versions of the canary runtimes.

syn-nodejs-2.0-beta

> **Important**
>
> This runtime version was deprecated on February 8, 2021. For more information, see [CloudWatch Synthetics runtime support policy](#).

Major dependencies:

- Lambda runtime Node.js 10.x
- Puppeteer-core version 3.3.0
- Chromium version 83.0.4103.0

Changes in syn-nodejs-2.0-beta:

- **Upgraded dependencies**— This runtime version uses Puppeteer-core version 3.3.0 and Chromium version 83.0.4103.0
- **Synthetics reporting**— For each canary run, CloudWatch Synthetics creates a report named `SyntheticsReport-PASSED.json` or `SyntheticsReport-FAILED.json` which records data such as start time, end time, status, and failures. It also records the PASSED/FAILED status of each step of the canary script, and failures and screenshots captured for each step.

- **Broken link checker report**— The new version of the broken link checker included in this runtime creates a report that includes the links that were checked, status code, failure reason (if any), and source and destination page screenshots.
- **New CloudWatch metrics**— Synthetics publishes metrics named `2xx`, `4xx`, `5xx`, and `RequestFailed` in the `CloudWatchSynthetics` namespace. These metrics show the number of 200s, 400s, 500s, and request failures in the canary runs. These metrics are reported only for UI canaries, and are not reported for API canaries.
- **Sortable HAR files**— You can now sort your HAR files by status code, request size, and duration.
- **Metrics timestamp**— CloudWatch metrics are now reported based on the Lambda invocation time instead of the canary run end time.

Bug fixes in syn-nodejs-2.0-beta:

- Fixed the issue of canary artifact upload errors not being reported. Such errors are now surfaced as execution errors.
- Fixed the issue of redirected requests (3xx) being incorrectly logged as errors.
- Fixed the issue of screenshots being numbered starting from 0. They should now start with 1.
- Fixed the issue of screenshots being garbled for Chinese and Japanese fonts.

syn-1.0

The first Synthetics runtime version is `syn-1.0`.

Major dependencies:

- Lambda runtime Node.js 10.x
- Puppeteer-core version 1.14.0
- The Chromium version that matches Puppeteer-core 1.14.0

Runtime versions using Python and Selenium Webdriver

The following sections contain information about the CloudWatch Synthetics runtime versions for Python and Selenium Webdriver. Selenium is an open-source browser automation tool. For more information about Selenium, see www.selenium.dev/

The naming convention for these runtime versions is `syn-language-framework-majorversion.minorversion`.

syn-python-selenium-5.1

Version 5.1 is the newest CloudWatch Synthetics runtime for Python and Selenium.

Major dependencies:

- Python 3.9
- Selenium 4.21.0
- Chromium version 131.0.6778.264

Changes in syn-python-selenium-5.1

- Minor updates on metric emission.
- Supports dry runs for the canary which allows for adhoc executions or performing a safe canary update.

Previous runtime versions for Python and Selenium

The following earlier runtime versions for Python and Selenium are still supported.

syn-python-selenium-5.0

Version 5.0 is the newest CloudWatch Synthetics runtime for Python and Selenium.

Major dependencies:

- Python 3.9
- Selenium 4.21.0
- Chromium version 131.0.6778.264

Changes in syn-python-selenium-5.0:

- Automatic retry if the browser fails to launch.

syn-python-selenium-4.1

Version 4.1 is the newest CloudWatch Synthetics runtime for Python and Selenium.

Major dependencies:

- Python 3.9
- Selenium 4.15.1
- Chromium version 126.0.6478.126

Changes in syn-python-selenium-4.1:

- **Addresses security vulnerability**– This runtime has an update to address the CVE-2024-39689 vulnerability.

syn-python-selenium-4.0

Major dependencies:

- Python 3.9
- Selenium 4.15.1
- Chromium version 126.0.6478.126

Changes in syn-python-selenium-4.0:

- **Bug fixes** for errors in HAR parser logging.

syn-python-selenium-3.0

Major dependencies:

- Python 3.8
- Selenium 4.15.1
- Chromium version 121.0.6167.139

Changes in syn-python-selenium-3.0:

- **Updated versions of the bundled libraries in Chromium**— The Chromium dependency is updated to a new version.

syn-python-selenium-2.1

Major dependencies:

- Python 3.8
- Selenium 4.15.1
- Chromium version 111.0.5563.146

Changes in syn-python-selenium-2.1:

- **Updated versions of the bundled libraries in Chromium**— The Chromium and Selenium dependencies are updated to new versions.

Deprecated runtime versions for Python and Selenium

The following earlier runtime versions for Python and Selenium have been deprecated. For information about runtime deprecation dates, see CloudWatch Synthetics runtime deprecation dates.

syn-python-selenium-2.0

Major dependencies:

- Python 3.8
- Selenium 4.10.0
- Chromium version 111.0.5563.146

Changes in syn-python-selenium-2.0:

- **Updated dependencies**— The Chromium and Selenium dependencies are updated to new versions.

Bug fixes in syn-python-selenium-2.0:

- **Timestamp added**— A timestamp has been added to canary logs.

- **Session re-use**— A bug was fixed so that canaries are now prevented from reusing the session from their previous canary run.

syn-python-selenium-1.3

Major dependencies:

- Python 3.8
- Selenium 3.141.0
- Chromium version 92.0.4512.0

Changes in syn-python-selenium-1.3:

- **More precise timestamps**— The start time and stop time of canary runs are now precise to the millisecond.

syn-python-selenium-1.2

Major dependencies:

- Python 3.8
- Selenium 3.141.0
- Chromium version 92.0.4512.0

- **Updated dependencies**— The only new features in this runtime are the updated dependencies.

syn-python-selenium-1.1

Major dependencies:

- Python 3.8
- Selenium 3.141.0
- Chromium version 83.0.4103.0

Features:

- **Custom handler function**— You can now use a custom handler function for your canary scripts. Previous runtimes required the script entry point to include `.handler`.

  You can also put canary scripts in any folder and pass the folder name as part of the handler. For example, `MyFolder/MyScriptFile.functionname` can be used as an entry point.
- **Configuration options for adding metrics and step failure configurations**— These options were already available in runtimes for Node.js canaries. For more information, see SyntheticsConfiguration class.
- **Custom arguments in Chrome** — You can now open a browser in incognito mode or pass in proxy server configuration. For more information, see Chrome().
- **Cross-Region artifact buckets**— A canary can store its artifacts in an Amazon S3 bucket in a different Region.
- **Bug fixes, including a fix for the `index.py` issue**— With previous runtimes, a canary file named `index.py` caused exceptions because it conflicted with the name of the library file. This issue is now fixed.

syn-python-selenium-1.0

Major dependencies:

- Python 3.8
- Selenium 3.141.0
- Chromium version 83.0.4103.0

Features:

- **Selenium support**— You can write canary scripts using the Selenium test framework. You can bring your Selenium scripts from elsewhere into CloudWatch Synthetics with minimal changes, and they will work with Amazon services.

Writing a canary script

The following sections explain how to write a canary script and how to integrate a canary with other Amazon services and with external dependencies and libraries.

Topics

Writing a Node.js canary script using the Playwright runtime

Topics

Packaging your Node.js canary files for the Playwright runtime

Your canary script comprises of a `.js` (CommonJS syntax) or `.mjs` (ES syntax) file containing your Synthetics handler code, together with any additional packages and modules your code depends on. Scripts created in ES (ECMAScript) format should either use .mjs as the extension or include a package.json file with the "type": "module" field set. Unlike other runtimes like Node.js Puppeteer, you are not required to save your scripts in a specific folder structure. You can package your scripts directly. Use your preferred `zip` utility to create a `.zip` file with your handler file at the root. If your canary script depends on additional packages or modules that aren't included in the Synthetics runtime, you can add these dependencies to your `.zip` file. To do so, you can install your function's required libraries in the `node_modules` directory by running the `npm install` command. The following example CLI commands create a `.zip` file named `my_deployment_package.zip` containing the `index.js` or `index.mjs` file (Synthetics handler) and its dependencies. In the example, you install dependencies using the npm package manager.

```
~/my_function
### index.mjs
### synthetics.json
### myhelper-util.mjs
### node_modules
    ### mydependency
```

Create a .zip file that contains the contents of your project folder at the root. Use the r (recursive) option, as shown in the following example, to ensure that zip compresses the subfolders.

```
zip -r my_deployment_package.zip .
```

Add a Synthetics configuration file to configure the behavior of CloudWatch Synthetics. You can create a synthetics.json file and save it at the same path as your entry point or handler file.

Optionally, you can also store your entry point file in a folder structure of your choice. However, be sure that the folder path is specified in your handler name.

Handler name

Be sure to set your canary's script entry point (handler) as myCanaryFilename.functionName to match the file name of your script's entry point. You can optionally store the canary in a separate folder such as myFolder/my_canary_filename.mjs. If you store it in a separate folder, specify that path in your script entry point, such as myFolder/my_canary_filename.functionName.

Changing an existing Playwright script to use as a CloudWatch Synthetics canary

You can edit an existing script for Node.js and Playwright to be used as a canary. For more information about Playwright, see the Playwright library documentation.

You can use the following Playwright script that is saved in file exampleCanary.mjs.

```
import { chromium } from 'playwright';
import { expect } from '@playwright/test';

const browser = await chromium.launch();
const page = await browser.newPage();
await page.goto('https://example.com', {timeout: 30000});
await page.screenshot({path: 'example-home.png'});

const title = await page.title();
expect(title).toEqual("Example Domain");

await browser.close();
```

Convert the script by performing the following steps:

1. Create and export a handler function. The handler is the entry point function for the script. You can choose any name for the handler function, but the function that is

used in your script should be the same as in your canary handler. If your script name is `exampleCanary.mjs`, and the handler function name is `myhandler`, your canary handler is named `exampleCanary.myhandler`. In the following example, the handler function name is `handler`.

```
exports.handler = async () => {
  // Your script here
  };
```

2. Import the `Synthetics Playwright module` as a dependency.

```
import { synthetics } from '@amzn/synthetics-playwright';
```

3. Launch a browser using the Synthetics `Launch` function.

```
const browser = await synthetics.launch();
```

4. Create a new Playwright page by using the Synthetics `newPage` function.

```
const page = await synthetics.newPage();
```

Your script is now ready to be run as a Synthetics canary. The following is the the updated script:

Updated script in ES6 format

The script file saved with a `.mjs` extension.

```
import { synthetics } from '@amzn/synthetics-playwright';
import { expect } from '@playwright/test';

export const handler = async (event, context) => {
  try {
        // Launch a browser
        const browser = await synthetics.launch();

        // Create a new page
        const page = await synthetics.newPage(browser);

        // Navigate to a website
        await page.goto('https://www.example.com', {timeout: 30000});
```

```
        // Take screenshot
        await page.screenshot({ path: '/tmp/example.png' });

        // Verify the page title
        const title = await page.title();
        expect(title).toEqual("Example Domain");
    } finally {
        // Ensure browser is closed
        await synthetics.close();
    }
};
```

Updated script in CommonJS format

The script file saved with a `.js` extension.

```
const { synthetics } = require('@amzn/synthetics-playwright');
const { expect } = require('@playwright/test');

exports.handler = async (event) => {
  try {
    const browser = await synthetics.launch();
    const page = await synthetics.newPage(browser);
    await page.goto('https://www.example.com', {timeout: 30000});
    await page.screenshot({ path: '/tmp/example.png' });
    const title = await page.title();
    expect(title).toEqual("Example Domain");
  } finally {
    await synthetics.close();
  }
};
```

CloudWatch Synthetics configurations

You can configure the behavior of the Synthetics Playwright runtime by providing an optional JSON configuration file named `synthetics.json`. This file should be packaged in the same location as the handler file. Though a configuration file is optional, f you don't provide a configuration file, or a configuration key is missing, CloudWatch assumes defaults.

Packaging your configuration file

The following are supported configuration values, and their defaults.

```
{
    "step": {
        "screenshotOnStepStart": false,
        "screenshotOnStepSuccess": false,
        "screenshotOnStepFailure": false,
        "stepSuccessMetric": true,
        "stepDurationMetric": true,
        "continueOnStepFailure": true,
        "stepsReport": true
    },
    "report": {
        "includeRequestHeaders": true,
        "includeResponseHeaders": true,
        "includeUrlPassword": false,
        "includeRequestBody": true,
        "includeResponseBody": true,
        "restrictedHeaders": ['x-amz-security-token', 'Authorization'], // Value of
 these headers is redacted from logs and reports
        "restrictedUrlParameters": ['Session', 'SigninToken'] // Values of these url
 parameters are redacted from logs and reports
    },
    "logging": {
        "logRequest": false,
        "logResponse": false,
        "logResponseBody": false,
        "logRequestBody": false,
        "logRequestHeaders": false,
        "logResponseHeaders": false
    },
    "httpMetrics": {
        "metric_2xx": true,
        "metric_4xx": true,
        "metric_5xx": true,
        "failedRequestsMetric": true,
        "aggregatedFailedRequestsMetric": true,
        "aggregated2xxMetric": true,
        "aggregated4xxMetric": true,
        "aggregated5xxMetric": true
    },
    "canaryMetrics": {
        "failedCanaryMetric": true,
        "aggregatedFailedCanaryMetric": true
    },
```

```
    "userAgent": "",
    "har": true
}
```

Step configurations

- `screenshotOnStepStart` – Determines if Synthetics should capture a screenshot before the step starts. The default is `true`.
- `screenshotOnStepSuccess` – Determines if Synthetics should capture a screenshot after a step has succeeded. The default is `true`.
- `screenshotOnStepFailure` – Determines if Synthetics should capture a screenshot after a step has failed. The default is `true`.
- `continueOnStepFailure` – Determines if a script should continue even after a step has failed. The default is `false`.
- `stepSuccessMetric` – Determines if a step's `SuccessPercent` metric is emitted. The `SuccessPercent` metric for a step is `100` for the canary run if the step succeeds, and `0` if the step fails. The default is `true`.
- `stepDurationMetric` – Determines if a step's `Duration` metric is emitted. The `Duration` metric is emitted as a duration, in milliseconds, of the step's run. The default is `true`.

Report configurations

Includes all reports generated by CloudWatch Synthetics, such as a HAR file and a Synthetics steps report. Sensitive data redaction fields `restrictedHeaders` and `restrictedUrlParameters` also apply to logs generated by Synthetics.

- `includeRequestHeaders` – Whether to include request headers in the report. The default is `false`.
- `includeResponseHeaders` – Whether to include response headers in the report. The default is `false`.
- `includeUrlPassword` – Whether to include a password that appears in the URL. By default, passwords that appear in URLs are redacted from logs and reports, to prevent the disclosure of sensitive data. The default is `false`.
- `includeRequestBody` – Whether to include the request body in the report. The default is `false`.

- `includeResponseBody` – Whether to include the response body in the report. The default is `false`.
- `restrictedHeaders` – A list of header values to ignore, if headers are included. This applies to both request and response headers. For example, you can hide your credentials by passing `includeRequestHeaders` as true and `restrictedHeaders` as `['Authorization']`.
- `restrictedUrlParameters` – A list of URL path or query parameters to redact. This applies to URLs that appear in logs, reports, and errors. The parameter is case-insensitive. You can pass an asterisk (*) as a value to redact all URL path and query parameter values. The default is an empty array.
- `har` – Determines if an HTTP archive (HAR) should be generated. The default is `true`.

The following is an example of a report configurations file.

```
"includeRequestHeaders": true,
"includeResponseHeaders": true,
"includeUrlPassword": false,
"includeRequestBody": true,
"includeResponseBody": true,
"restrictedHeaders": ['x-amz-security-token', 'Authorization'], // Value of these
 headers is redacted from logs and reports
"restrictedUrlParameters": ['Session', 'SigninToken'] // Values of these URL parameters
 are redacted from logs and reports
```

Logging configurations

Applies to logs generated by CloudWatch Synthetics. Controls the verbosity of request and response logs.

- `logRequest` – Whether to log every request in canary logs. For UI canaries, this logs each request sent by the browser. The default is `false`.
- `logResponse` – Whether to log every response in canary logs. For UI canaries, this logs every response received by the browser. The default is `false`.
- `logRequestBody` – Whether to log request bodies along with the requests in canary logs. This configuration applies only if `logRequest` is true. The default is `false`.
- `logResponseBody` – Whether to log response bodies along with the requests in canary logs. This configuration applies only if `logResponse` is true. The default is `false`.

- `logRequestHeaders` – Whether to log request headers along with the requests in canary logs. This configuration applies only if `logRequest` is true. The default is `false`.
- `logResponseHeaders` – Whether to log response headers along with the responses in canary logs. This configuration applies only if `logResponse` is true. The default is `false`.

HTTP metric configurations

Configurations for metrics related to the count of network requests with different HTTP status codes, emitted by CloudWatch Synthetics for this canary.

- `metric_2xx` – Whether to emit the `2xx` metric (with the `CanaryName` dimension) for this canary. The default is `true`.
- `metric_4xx` – Whether to emit the `4xx` metric (with the `CanaryName` dimension) for this canary. The default is `true`.
- `metric_5xx` – Whether to emit the `5xx` metric (with the `CanaryName` dimension) for this canary. The default is `true`.
- `failedRequestsMetric` – Whether to emit the `failedRequests` metric (with the `CanaryName` dimension) for this canary. The default is `true`.
- `aggregatedFailedRequestsMetric` – Whether to emit the `failedRequests` metric (without the `CanaryName` dimension) for this canary. The default is `true`.
- `aggregated2xxMetric` – Whether to emit the `2xx` metric (without the `CanaryName` dimension) for this canary. The default is `true`.
- `aggregated4xxMetric` – Whether to emit the `4xx` metric (without the `CanaryName` dimension) for this canary. The default is `true`.
- `aggregated5xxMetric` – Whether to emit the `5xx` metric (without the `CanaryName` dimension) for this canary. The default is `true`.

Canary metric configurations

Configurations for other metrics emitted by CloudWatch Synthetics.

- `failedCanaryMetric` – Whether to emit the `Failed` metric (with the `CanaryName` dimension) for this canary. The default is `true`.
- `aggregatedFailedCanaryMetric` – Whether to emit the `Failed` metric (without the `CanaryName` dimension) for this canary. The default is `true`.

Other configurations

- `userAgent` – A string to append to the user agent. The user agent is a string that is included in request header, and identifies your browser to websites you visit when you use the headless browser. CloudWatch Synthetics automatically adds `CloudWatchSynthetics/canary-arn to the user agent`. The specified configuration is appended to the generated user agent. The default user agent value to append is an empty string ("").

CloudWatch Synthetics environment variables

Configure the logging level and format by using environment variables.

Log format

The CloudWatch Synthetics Playwright runtime creates CloudWatch logs for every canary run. Logs are written in JSON format for convenient querying. Optionally, you can change the log format to `TEXT`.

- `Environment variable name` – CW_SYNTHETICS_LOG_FORMAT
- `Supported values` – JSON, TEXT
- `Default` – JSON

Log levels

Though enabling `Debug` mode increases verbosity, it can be useful for troubleshooting.

- `Environment variable name` – CW_SYNTHETICS_LOG_LEVEL
- `Supported values` – TRACE, DEBUG, INFO, WARN, ERROR, FATAL
- `Default` – INFO

Writing a Node.js canary script using the Puppeteer runtime

Topics

- [Integrating your canary with other Amazon services]
- [Forcing your canary to use a static IP address]

Creating a CloudWatch Synthetics canary from scratch

Here is an example minimal Synthetics Canary script. This script passes as a successful run, and returns a string. To see what a failing canary looks like, change `let fail = false;` to `let fail = true;`.

You must define an entry point function for the canary script. To see how files are uploaded to the Amazon S3 location specified as the canary's `ArtifactS3Location`, create these files in the `/tmp` folder. All canary artifacts should be stored in `/tmp`, because it's the only writable directory. Be sure that the screenshot path is set to `/tmp` for any screenshots or other files created by the script. Synthetics automatically uploads files in `/tmp` to an S3 bucket.

```
/tmp/<name>
```

After the script runs, the pass/fail status and the duration metrics are published to CloudWatch and the files under `/tmp` are uploaded to an S3 bucket.

```
const basicCustomEntryPoint = async function () {

    // Insert your code here

    // Perform multi-step pass/fail check

    // Log decisions made and results to /tmp

    // Be sure to wait for all your code paths to complete 
    // before returning control back to Synthetics.
    // In that way, your canary will not finish and report success
    // before your code has finished executing

    // Throw to fail, return to succeed
    let fail = false;
    if (fail) {
        throw "Failed basicCanary check.";
    }

    return "Successfully completed basicCanary checks.";
};
```

```
exports.handler = async () => {
    return await basicCustomEntryPoint();
};
```

Next, we'll expand the script to use Synthetics logging and make a call using the Amazon SDK. For demonstration purposes, this script will create an Amazon DynamoDB client and make a call to the DynamoDB listTables API. It logs the response to the request and logs either pass or fail depending on whether the request was successful.

```
const log = require('SyntheticsLogger');
const AWS = require('aws-sdk');
// Require any dependencies that your script needs
// Bundle additional files and dependencies into a .zip file with folder structure
// nodejs/node_modules/additional files and folders

const basicCustomEntryPoint = async function () {

    log.info("Starting DynamoDB:listTables canary.");

    let dynamodb = new AWS.DynamoDB();
    var params = {};
    let request = await dynamodb.listTables(params);
    try {
        let response = await request.promise();
        log.info("listTables response: " + JSON.stringify(response));
    } catch (err) {
        log.error("listTables error: " + JSON.stringify(err), err.stack);
        throw err;
    }

    return "Successfully completed DynamoDB:listTables canary.";
};

exports.handler = async () => {
    return await basicCustomEntryPoint();
};
```

Packaging your Node.js canary files

If you are uploading your canary scripts using an Amazon S3 location, your zip file should include your script under this folder structure: `nodejs/node_modules/`*`myCanaryFilename.js file`*.

If you have more than a single `.js` file or you have a dependency that your script depends on, you can bundle them all into a single ZIP file that contains the folder structure `nodejs/node_modules/`*`myCanaryFilename.js file and other folders and files`*. If you are using `syn-nodejs-puppeteer-3.4` or later, you can optionally put your canary files in another folder and creating your folder structure like this: `nodejs/node_modules/`*`myFolder`*`/`*`myCanaryFilename.js file and other folders and files`*.

Handler name

Be sure to set your canary's script entry point (handler) as `myCanaryFilename.functionName` to match the file name of your script's entry point. If you are using a runtime earlier than `syn-nodejs-puppeteer-3.4`, then `functionName` must be `handler`. If you are using `syn-nodejs-puppeteer-3.4` or later, you can choose any function name as the handler. If you are using `syn-nodejs-puppeteer-3.4` or later, you can also optionally store the canary in a separate folder such as `nodejs/node_modules/myFolder/my_canary_filename`. If you store it in a separate folder, specify that path in your script entry point, such as `myFolder/my_canary_filename.functionName`.

Changing an existing Puppeteer script to use as a Synthetics canary

This section explains how to take Puppeteer scripts and modify them to run as Synthetics canary scripts. For more information about Puppeteer, see Puppeteer API v1.14.0.

We'll start with this example Puppeteer script:

```
const puppeteer = require('puppeteer');

(async () => {
  const browser = await puppeteer.launch();
  const page = await browser.newPage();
  await page.goto('https://example.com');
  await page.screenshot({path: 'example.png'});

  await browser.close();
})();
```

The conversion steps are as follows:

- Create and export a `handler` function. The handler is the entry point function for the script. If you are using a runtime earlier than `syn-nodejs-puppeteer-3.4`, the handler function must be named `handler`. If you are using `syn-nodejs-puppeteer-3.4` or later, the function can have any name, but it must be the same name that is used in the script. Also, if you are using `syn-nodejs-puppeteer-3.4` or later, you can store your scripts under any folder and specify that folder as part of the handler name.

```
const basicPuppeteerExample = async function () {};

exports.handler = async () => {
    return await basicPuppeteerExample();
};
```

- Use the `Synthetics` dependency.

```
var synthetics = require('Synthetics');
```

- Use the `Synthetics.getPage` function to get a Puppeteer `Page` object.

```
const page = await synthetics.getPage();
```

The page object returned by the Synthetics.getPage function has the **page.on** `request`, `response` and `requestfailed` events instrumented for logging. Synthetics also sets up HAR file generation for requests and responses on the page, and adds the canary ARN to the user-agent headers of outgoing requests on the page.

The script is now ready to be run as a Synthetics canary. Here is the updated script:

```
var synthetics = require('Synthetics');  // Synthetics dependency

const basicPuppeteerExample = async function () {
    const page = await synthetics.getPage(); // Get instrumented page from Synthetics
    await page.goto('https://example.com');
    await page.screenshot({path: '/tmp/example.png'}); // Write screenshot to /tmp
 folder
};

exports.handler = async () => {  // Exported handler function
    return await basicPuppeteerExample();
};
```

Environment variables

You can use environment variables when creating canaries. This allows you to write a single canary script and then use that script with different values to quickly create multiple canaries that have a similar task.

For example, suppose your organization has endpoints such as `prod`, `dev`, and `pre-release` for the different stages of your software development, and you need to create canaries to test each of these endpoints. You can write a single canary script that tests your software and then specify different values for the endpoint environment variable when you create each of the three canaries. Then, when you create a canary, you specify the script and the values to use for the environment variables.

The names of environment variables can contain letters, numbers, and the underscore character. They must start with a letter and be at least two characters. The total size of your environment variables can't exceed 4 KB. You can't specify any Lambda reserved environment variables as the names of your environment variables. For more information about reserved environment variables, see [Runtime environment variables](#).

> **Important**
>
> Environment variable keys and values are encrypted at rest using Amazon owned Amazon KMS keys. However, the environment variables are not encrypted on the client side. Do not store sensitive information in them.

The following example script uses two environment variables. This script is for a canary that checks whether a webpage is available. It uses environment variables to parameterize both the URL that it checks and the CloudWatch Synthetics log level that it uses.

The following function sets `LogLevel` to the value of the `LOG_LEVEL` environment variable.

```
synthetics.setLogLevel(process.env.LOG_LEVEL);
```

This function sets `URL` to the value of the `URL` environment variable.

```
const URL = process.env.URL;
```

This is the complete script. When you create a canary using this script, you specify values for the LOG_LEVEL and URL environment variables.

```
var synthetics = require('Synthetics');
const log = require('SyntheticsLogger');

const pageLoadEnvironmentVariable = async function () {

    // Setting the log level (0-3)
    synthetics.setLogLevel(process.env.LOG_LEVEL);
    // INSERT URL here
    const URL = process.env.URL;

    let page = await synthetics.getPage();
    //You can customize the wait condition here. For instance,
    //using 'networkidle2' may be less restrictive.
    const response = await page.goto(URL, {waitUntil: 'domcontentloaded', timeout:
 30000});
    if (!response) {
        throw "Failed to load page!";
    }
    //Wait for page to render.
    //Increase or decrease wait time based on endpoint being monitored.
    await page.waitFor(15000);
    await synthetics.takeScreenshot('loaded', 'loaded');
    let pageTitle = await page.title();
    log.info('Page title: ' + pageTitle);
    log.debug('Environment variable:' + process.env.URL);

    //If the response status code is not a 2xx success code
    if (response.status() < 200 || response.status() > 299) {
        throw "Failed to load page!";
    }
};

exports.handler = async () => {
    return await pageLoadEnvironmentVariable();
};
```

Passing environment variables to your script

To pass environment variables to your script when you create a canary in the console, specify the keys and values of the environment variables in the **Environment variables** section on the console. For more information, see Creating a canary.

To pass environment variables through the API or Amazon CLI, use the `EnvironmentVariables` parameter in the `RunConfig` section. The following is an example Amazon CLI command that creates a canary that uses two environment variables with keys of `Environment` and `Region`.

```
aws synthetics create-canary --cli-input-json '{
   "Name":"nameofCanary",
   "ExecutionRoleArn":"roleArn",
   "ArtifactS3Location":"s3://amzn-s3-demo-bucket-123456789012-us-west-2",
   "Schedule":{
      "Expression":"rate(0 minute)",
      "DurationInSeconds":604800
   },
   "Code":{
      "S3Bucket": "canarycreation",
      "S3Key": "cwsyn-mycanaryheartbeat-12345678-d1bd-1234-
abcd-123456789012-12345678-6a1f-47c3-b291-123456789012.zip",
      "Handler":"pageLoadBlueprint.handler"
   },
   "RunConfig": {
      "TimeoutInSeconds":60,
      "EnvironmentVariables": {
         "Environment":"Production",
         "Region": "us-west-1"
      }
   },
   "SuccessRetentionPeriodInDays":13,
   "FailureRetentionPeriodInDays":13,
   "RuntimeVersion":"syn-nodejs-2.0"
}'
```

Integrating your canary with other Amazon services

All canaries can use the Amazon SDK library. You can use this library when you write your canary to integrate the canary with other Amazon services.

To do so, you need to add the following code to your canary. For these examples, Amazon Secrets Manager is used as the service that the canary is integrating with.

- Import the Amazon SDK.

```
const AWS = require('aws-sdk');
```

- Create a client for the Amazon service that you are integrating with.

```
const secretsManager = new AWS.SecretsManager();
```

- Use the client to make API calls to that service.

```
var params = {
  SecretId: secretName
};
return await secretsManager.getSecretValue(params).promise();
```

The following canary script code snippet demonstrates an example of integration with Secrets Manager in more detail.

```
var synthetics = require('Synthetics');
const log = require('SyntheticsLogger');

const AWS = require('aws-sdk');
const secretsManager = new AWS.SecretsManager();

const getSecrets = async (secretName) => {
    var params = {
        SecretId: secretName
    };
    return await secretsManager.getSecretValue(params).promise();
}

const secretsExample = async function () {
    let URL = "<URL>";
    let page = await synthetics.getPage();

    log.info(`Navigating to URL: ${URL}`);
    const response = await page.goto(URL, {waitUntil: 'domcontentloaded', timeout:
 30000});

    // Fetch secrets
    let secrets = await getSecrets("secretname")
```

```
    /**
    * Use secrets to login.
    *
    * Assuming secrets are stored in a JSON format like:
    * {
    *   "username": "<USERNAME>",
    *   "password": "<PASSWORD>"
    * }
    **/
    let secretsObj = JSON.parse(secrets.SecretString);
    await synthetics.executeStep('login', async function () {
        await page.type(">USERNAME-INPUT-SELECTOR<", secretsObj.username);
        await page.type(">PASSWORD-INPUT-SELECTOR<", secretsObj.password);

        await Promise.all([
          page.waitForNavigation({ timeout: 30000 }),
          await page.click(">SUBMIT-BUTTON-SELECTOR<")
        ]);
    });

    // Verify login was successful
    await synthetics.executeStep('verify', async function () {
        await page.waitForXPath(">SELECTOR<", { timeout: 30000 });
    });
};

exports.handler = async () => {
    return await secretsExample();
};
```

Forcing your canary to use a static IP address

You can set up a canary so that it uses a static IP address.

To force a canary to use a static IP address

1. Create a new VPC. For more information, see Using DNS with Your VPC.
2. Create a new internet gateway. For more information, see Adding an internet gateway to your VPC.
3. Create a public subnet inside your new VPC.
4. Add a new route table to the VPC.

5. Add a route in the new route table, that goes from `0.0.0.0/0` to the internet gateway.
6. Associate the new route table with the public subnet.
7. Create an elastic IP address. For more information, see Elastic IP addresses .
8. Create a new NAT gateway and assign it to the public subnet and the elastic IP address.
9. Create a private subnet inside the VPC.
10. Add a route to the VPC default route table, that goes from `0.0.0.0/0` to the NAT gateway
11. Create your canary.

Writing a Python canary script

This script passes as a successful run, and returns a string. To see what a failing canary looks like, change fail = False to fail = True

```
def basic_custom_script():
    # Insert your code here
    # Perform multi-step pass/fail check
    # Log decisions made and results to /tmp
    # Be sure to wait for all your code paths to complete
    # before returning control back to Synthetics.
    # In that way, your canary will not finish and report success
    # before your code has finished executing
    fail = False
    if fail:
        raise Exception("Failed basicCanary check.")
    return "Successfully completed basicCanary checks."
def handler(event, context):
    return basic_custom_script()
```

Packaging your Python canary files

If you have more than one .py file or your script has a dependency, you can bundle them all into a single ZIP file. If you use the `syn-python-selenium-1.1` runtime, the ZIP file must contain your main canary .py file within a `python` folder, such as `python/my_canary_filename.py`. If you use `syn-python-selenium-1.1` or later, you can optionally use a different folder , such as `python/myFolder/my_canary_filename.py`.

This ZIP file should contain all necessary folders and files, but the other files do not need to be in the `python` folder.

Be sure to set your canary's script entry point as `my_canary_filename.functionName` to match the file name and function name of your script's entry point. If you are using the `syn-python-selenium-1.0` runtime, then `functionName` must be `handler`. If you are using `syn-python-selenium-1.1` or later, this handler name restriction doesn't apply, and you can also optionally store the canary in a separate folder such as `python/myFolder/my_canary_filename.py`. If you store it in a separate folder, specify that path in your script entry point, such as `myFolder/my_canary_filename.functionName`.

Changing an existing Selenium script to use a Synthetics canary

You can quickly modify an existing script for Python and Selenium to be used as a canary. For more information about Selenium, see www.selenium.dev/.

For this example, we'll start with the following Selenium script:

```
from selenium import webdriver

def basic_selenium_script():
    browser = webdriver.Chrome()
    browser.get('https://example.com')
    browser.save_screenshot('loaded.png')

basic_selenium_script()
```

The conversion steps are as follows.

To convert a Selenium script to be used as a canary

1. Change the `import` statement to use Selenium from the `aws_synthetics` module:

   ```
   from aws_synthetics.selenium import synthetics_webdriver as webdriver
   ```

   The Selenium module from `aws_synthetics` ensures that the canary can emit metrics and logs, generate a HAR file, and work with other CloudWatch Synthetics features.
2. Create a handler function and call your Selenium method. The handler is the entry point function for the script.

   If you are using `syn-python-selenium-1.0`, the handler function must be named `handler`. If you are using `syn-python-selenium-1.1` or later, the function can have any name, but it must be the same name that is used in the script. Also, if you are using `syn-python-`

selenium-1.1 or later, you can store your scripts under any folder and specify that folder as part of the handler name.

```
def handler(event, context):
    basic_selenium_script()
```

The script is now updated to be a CloudWatch Synthetics canary. Here is the updated script:

```
from aws_synthetics.selenium import synthetics_webdriver as webdriver

def basic_selenium_script():
    browser = webdriver.Chrome()
    browser.get('https://example.com')
    browser.save_screenshot('loaded.png')

def handler(event, context):
    basic_selenium_script()
```

Changing an existing Puppeteer Synthetics script to authenticate non-standard certificates

One important use case for Synthetics canaries is for you to monitor your own endpoints. If you want to monitor an endpoint that isn't ready for external traffic, this monitoring can sometimes mean that you don't have a proper certificate signed by a trusted third-party certificate authority.

Two possible solutions to this scenario are as follows:

- To authenticate a client certificate, see How to validate authentication using Amazon CloudWatch Synthetics – Part 2.
- To authenticate a self-signed certificate, see How to validate authentication with self-signed certificates in Amazon CloudWatch Synthetics

You are not limited to these two options when you use CloudWatch Synthetics canaries. You can extend these features and add your business logic by extending the canary code.

> **Note**
>
> Synthetics canaries running on Python runtimes innately have the `--ignore-certificate-errors` flag enabled, so those canaries shouldn't have any issues reaching sites with non-standard certificate configurations.

Library functions available for canary scripts

CloudWatch Synthetics includes several built-in classes and functions that you can call when writing Node.js scripts to use as canaries.

Some apply to both UI and API canaries. Others apply to UI canaries only. A UI canary is a canary that uses the `getPage()` function and uses Puppeteer as a web driver to navigate and interact with webpages.

> **Note**
>
> Whenever you upgrade a canary to use a new version of the the Synthetics runtime, all Synthetics library functions that your canary uses are also automatically upgraded to the same version of NodeJS that the Synthetics runtime supports.

Topics

Library functions available for Node.js canary scripts using Playwright

This section describes the library functions that are available for canary scripts using the Node.js Playwright runtime.

Topics

- getDefaultLaunchOptions
- executeStep

launch

This function launches a Chromium browser using a Playwright launch function, and returns the browser object. It decompresses browser binaries and launches the chromium browser by using default options suitable for a headless browser. For more information about the `launch` function, see `launch` in the Playwright documentation.

Usage

```
const browser = await synthetics.launch();
```

Arguments

`options` options (optional) is a configurable set of options for the browser.

Returns

Promise <Browser> where Browser is a Playwright browser instance.

If this function is called again, a previously-opened browser is closed before initiating a new browser. You can override launch parameters used by CloudWatch Synthetics, and pass additional parameters when launching the browser. For example, the following code snippet launches a browser with default arguments and a default executable path, but with a viewport of 800 x 600 pixels. For more information, see Playwright launch options in the Playwright documentation.

```
const browser = await synthetics.launch({
  defaultViewport: { 
      "deviceScaleFactor": 1, 
      "width": 800,
      "height": 600 
}});
```

You can also add or override Chromium flags passed on by default to the browser. For example, you can disable web security by adding a `--disable-web-security` flag to arguments in the CloudWatch Synthetics launch parameters:

```
// This function adds the --disable-web-security flag to the launch parameters
```

```
const defaultOptions = await synthetics.getDefaultLaunchOptions();
const launchArgs = [...defaultOptions.args, '--disable-web-security'];
const browser = await synthetics.launch({
    args: launchArgs
  });
```

newPage

The `newPage()` function creates and returns a new Playwright page. Synthetics automatically sets up a Chrome DevTools Protocol (CDP) connection to enable network captures for HTTP archive (HAR) generation.

Usage

Use `newPage()` in either of the following ways:

1. Creating a new page in a new browser context:

```
const page = await synthetics.newPage(browser);
```

2. Creating a new page in a specified browser context:

```
// Create a new browser context
const browserContext = await browser.newContext();

// Create a new page in the specified browser context
const page = await synthetics.newPage(browserContext)
```

Arguments

Accepts either Playwright [Browser](#) instance or Playwright [BrowserContext](#) instance.

Returns

Promise <Page> where Page is a Playwright [Page](#) instance.

close

Closes the currently-opened browser.

Usage

```
await synthetics.close();
```

It is recommended to close the browser in a `finally` block of your script.

Arguments

None

Returns

Returns Promise<void> used by the Synthetics launch function for launching the browser.

getDefaultLaunchOptions

The `getDefaultLaunchOptions()` function returns the browser launch options that are used by CloudWatch Synthetics.

Usage

```
const defaultOptions = await synthetics.getDefaultLaunchOptions();
```

Arguments

None

Returns

Returns Playwright [launch options](#) used by the Synthetics `launch` function for launching the browser.

executeStep

The `executeStep` function is used to execute a step in a Synthetics script. In CloudWatch Synthetics, a Synthetics step is a way to break down your canary script into a series of clearly defined actions, allowing you to monitor different parts of your application journey separately. For each step, CloudWatch Synthetics does the following:

- Automatically captures a screenshot before step starts and after a step is complete. You can also capture screenshots inside a step. Screenshots are captured by default, but can be turned off by using Synthetics configurations *`(Todo: Link)`*.
- A report, including a summary, of step execution details like the duration of a step, `pass` or `fail` status, source and destination page URLs, associated screenshots, etc. is created for each canary

run. When you choose a run in the CloudWatch Synthetics console, you can view execution details of each step on the **Step** tab.

- `SuccessPercent` and `Duration` CloudWatch metrics are emitted for each step, enabling users to monitor availability and latency of each step.

Usage

```
await synthetics.executeStep("mystepname", async function () {
  await page.goto(url, { waitUntil: 'load', timeout: 30000 });
}
```

> **Note**
>
> Steps should run sequentially. Be sure to use `await` on promises.

Arguments

- `stepName` string (required) (boolean)— Name of the Synthetics step.
- `functionToExecute` async function (required)— The function that you want Synthetics to run. This function should contain the logic for the step.
- `stepConfig` object (optional)— Step configuration overrides the global Synthetics configuration for this step.
  - `continueOnStepFailure` boolean (optional) — Whether to continue running the canary script after this step fails.
  - `screenshotOnStepStart` boolean (optional) — Whether to take a screenshot at the start of this step.
  - `screenshotOnStepSuccess` boolean (optional) — Whether to take a screenshot if this step succeeds.
  - `screenshotOnStepFailure` boolean (optional) — Whether to take a screenshot if this step fails.
- `page` — Playwright page object (optional)

  A Playwright page object. Synthetics uses this page object to capture screenshots and URLs. By default, Synthetics uses the Playwright page created when the `synthetics.newPage()` function is called for capturing page details like screenshots and URLs.

Returns

Returns a Promise that resolves with the value returned by the `functionToExecute` function. For an example script, see Sample code for canary scripts in this guide.

Library functions available for Node.js canary scripts using Puppeteer

This section describes the library functions available for Node.js canary scripts.

Topics

Node.js library classes and functions that apply to all canaries

The following CloudWatch Synthetics library functions for Node.js are useful for all canaries.

Topics

Synthetics class

The following functions for all canaries are in the Synthetics class.

Topics

- setLogLevel();

addExecutionError(errorMessage, ex);

`errorMessage` describes the error and `ex` is the exception that is encountered

You can use `addExecutionError` to set execution errors for your canary. It fails the canary without interrupting the script execution. It also doesn't impact your `successPercent` metrics.

You should track errors as execution errors only if they are not important to indicate the success or failure of your canary script.

An example of the use of `addExecutionError` is the following. You are monitoring the availability of your endpoint and taking screenshots after the page has loaded. Because the failure of taking a screenshot doesn't determine availability of the endpoint, you can catch any errors encountered while taking screenshots and add them as execution errors. Your availability metrics will still indicate that the endpoint is up and running, but your canary status will be marked as failed. The following sample code block catches such an error and adds it as an execution error.

```
try {
    await synthetics.takeScreenshot(stepName, "loaded");
} catch(ex) {
    synthetics.addExecutionError('Unable to take screenshot ', ex);
}
```

getCanaryName();

Returns the name of the canary.

getCanaryArn();

Returns the ARN of the canary.

getCanaryUserAgentString();

Returns the custom user agent of the canary.

getRuntimeVersion();

This function is available in runtime version `syn-nodejs-puppeteer-3.0` and later. It returns the Synthetics runtime version of the canary. For example, the return value could be `syn-nodejs-puppeteer-3.0`.

getLogLevel();

Retrieves the current log level for the Synthetics library. Possible values are the following:

- `0` – Debug
- `1` – Info
- `2` – Warn
- `3` – Error

Example:

```
let logLevel = synthetics.getLogLevel();
```

setLogLevel();

Sets the log level for the Synthetics library. Possible values are the following:

- `0` – Debug
- `1` – Info
- `2` – Warn
- `3` – Error

Example:

```
synthetics.setLogLevel(0);
```

SyntheticsConfiguration class

This class is available only in the `syn-nodejs-2.1` runtime version or later.

You can use the `SyntheticsConfiguration` class to configure the behavior of Synthetics library functions. For example, you can use this class to configure the `executeStep()` function to not capture screenshots.

You can set CloudWatch Synthetics configurations at the global level, which are applied to all steps of canaries. You can also override these configurations at the step level by passing configuration key and value pairs.

You can pass in options at the step level. For examples, see async executeStep(stepName, functionToExecute, [stepConfig]); and executeHttpStep(stepName, requestOptions, [callback], [stepConfig])

Topics

- setConfig(options)
- Visual monitoring

setConfig(options)

`options` is an object, which is a set of configurable options for your canary. The following sections explain the possible fields in *`options`*.

setConfig(options) for all canaries

For canaries using `syn-nodejs-puppeteer-3.2` or later, the **(options)** for **setConfig** can include the following parameters:

- `includeRequestHeaders` (boolean)— Whether to include request headers in the report. The default is `false`.
- `includeResponseHeaders` (boolean)— Whether to include response headers in the report. The default is `false`.
- `restrictedHeaders` (array)— A list of header values to ignore, if headers are included. This applies to both request and response headers. For example, you can hide your credentials by passing **includeRequestHeaders** as `true` and **restrictedHeaders** as `['Authorization']`.
- `includeRequestBody` (boolean)— Whether to include the request body in the report. The default is `false`.
- `includeResponseBody` (boolean)— Whether to include the response body in the report. The default is `false`.

  If you enable either `includeResponseBody` or `logResponseBody`, the data object is not returned in the response from some APIs, such as aws-sdk v3 clients. This is because of a limitation of Node.js and the type of response object used.

setConfig(options) regarding CloudWatch metrics

For canaries using `syn-nodejs-puppeteer-3.1` or later, the **(options)** for **setConfig** can include the following Boolean parameters that determine which metrics are published by the canary. The default for each of these options is `true`. The options that start with `aggregated` determine whether the metric is emitted without the `CanaryName` dimension. You can use these metrics to see the aggregated results for all of your canaries. The other options determine whether the metric is emitted with the `CanaryName` dimension. You can use these metrics to see results for each individual canary.

For a list of CloudWatch metrics emitted by canaries, see [CloudWatch metrics published by canaries](#).

- `failedCanaryMetric` (boolean)— Whether to emit the `Failed` metric (with the `CanaryName` dimension) for this canary. The default is `true`.
- `failedRequestsMetric` (boolean)— Whether to emit the `Failed requests` metric (with the `CanaryName` dimension) for this canary. The default is `true`.
- `_2xxMetric` (boolean)— Whether to emit the `2xx` metric (with the `CanaryName` dimension) for this canary. The default is `true`.
- `_4xxMetric` (boolean)— Whether to emit the `4xx` metric (with the `CanaryName` dimension) for this canary. The default is `true`.
- `_5xxMetric` (boolean)— Whether to emit the `5xx` metric (with the `CanaryName` dimension) for this canary. The default is `true`.
- `stepDurationMetric` (boolean)— Whether to emit the `Step duration` metric (with the `CanaryName` `StepName` dimensions) for this canary. The default is `true`.
- `stepSuccessMetric` (boolean)— Whether to emit the `Step success` metric (with the `CanaryName` `StepName` dimensions) for this canary. The default is `true`.
- `aggregatedFailedCanaryMetric` (boolean)— Whether to emit the `Failed` metric (without the `CanaryName` dimension) for this canary. The default is `true`.
- `aggregatedFailedRequestsMetric` (boolean)— Whether to emit the `Failed Requests` metric (without the `CanaryName` dimension) for this canary. The default is `true`.
- `aggregated2xxMetric` (boolean)— Whether to emit the `2xx` metric (without the `CanaryName` dimension) for this canary. The default is `true`.
- `aggregated4xxMetric` (boolean)— Whether to emit the `4xx` metric (without the `CanaryName` dimension) for this canary. The default is `true`.
- `aggregated5xxMetric` (boolean)— Whether to emit the `5xx` metric (without the `CanaryName` dimension) for this canary. The default is `true`.

- `visualMonitoringSuccessPercentMetric` (boolean)— Whether to emit the `visualMonitoringSuccessPercent` metric for this canary. The default is `true`.
- `visualMonitoringTotalComparisonsMetric` (boolean)— Whether to emit the `visualMonitoringTotalComparisons` metric for this canary. The default is `false`.
- `includeUrlPassword` (boolean)— Whether to include a password that appears in the URL. By default, passwords that appear in URLs are redacted from logs and reports, to prevent disclosing sensitive data. The default is `false`.
- `restrictedUrlParameters` (array)— A list of URL path or query parameters to redact. This applies to URLs appearing in logs, reports, and errors. The parameter is case-insensitive. You can pass an asterisk (*) as a value to redact all URL path and query parameter values. The default is an empty array.
- `logRequest` (boolean)— Whether to log every request in canary logs. For UI canaries, this logs each request sent by the browser. The default is `true`.
- `logResponse` (boolean)— Whether to log every response in canary logs. For UI canaries, this logs every response received by the browser. The default is `true`.
- `logRequestBody` (boolean)— Whether to log request bodies along with the requests in canary logs. This configuration applies only if `logRequest` is `true`. The default is `false`.
- `logResponseBody` (boolean)— Whether to log response bodies along with the responses in canary logs. This configuration applies only if `logResponse` is `true`. The default is `false`.

  If you enable either `includeResponseBody` or `logResponseBody`, the data object is not returned in the response from some APIs, such as aws-sdk v3 clients. This is because of a limitation of Node.js and the type of response object used.
- `logRequestHeaders` (boolean)— Whether to log request headers along with the requests in canary logs. This configuration applies only if `logRequest` is `true`. The default is `false`.

  Note that `includeRequestHeaders` enables headers in artifacts.
- `logResponseHeaders` (boolean)— Whether to log response headers along with the responses in canary logs. This configuration applies only if `logResponse` is `true`. The default is `false`.

  Note that `includeResponseHeaders` enables headers in artifacts.

> **Note**
>
> The `Duration` and `SuccessPercent` metrics are always emitted for each canary, both with and without the `CanaryName` metric.

Methods to enable or disable metrics

disableAggregatedRequestMetrics()

Disables the canary from emitting all request metrics that are emitted with no `CanaryName` dimension.

disableRequestMetrics()

Disables all request metrics, including both per-canary metrics and metrics aggregated across all canaries.

disableStepMetrics()

Disables all step metrics, including both step success metrics and step duration metrics.

enableAggregatedRequestMetrics()

Enables the canary to emit all request metrics that are emitted with no `CanaryName` dimension.

enableRequestMetrics()

Enables all request metrics, including both per-canary metrics and metrics aggregated across all canaries.

enableStepMetrics()

Enables all step metrics, including both step success metrics and step duration metrics.

get2xxMetric()

Returns whether the canary emits a `2xx` metric with the `CanaryName` dimension.

get4xxMetric()

Returns whether the canary emits a `4xx` metric with the `CanaryName` dimension.

get5xxMetric()

Returns whether the canary emits a `5xx` metric with the `CanaryName` dimension.

getAggregated2xxMetric()

Returns whether the canary emits a `2xx` metric with no dimension.

getAggregated4xxMetric()

Returns whether the canary emits a `4xx` metric with no dimension.

getAggregatedFailedCanaryMetric()

Returns whether the canary emits a `Failed` metric with no dimension.

getAggregatedFailedRequestsMetric()

Returns whether the canary emits a `Failed requests` metric with no dimension.

getAggregated5xxMetric()

Returns whether the canary emits a `5xx` metric with no dimension.

getFailedCanaryMetric()

Returns whether the canary emits a `Failed` metric with the `CanaryName` dimension.

getFailedRequestsMetric()

Returns whether the canary emits a `Failed requests` metric with the `CanaryName` dimension.

getStepDurationMetric()

Returns whether the canary emits a `Duration` metric with the `CanaryName` dimension for this canary.

getStepSuccessMetric()

Returns whether the canary emits a `StepSuccess` metric with the `CanaryName` dimension for this canary.

with2xxMetric(_2xxMetric)

Accepts a Boolean argument, which specifies whether to emit a `2xx` metric with the `CanaryName` dimension for this canary.

with4xxMetric(_4xxMetric)

Accepts a Boolean argument, which specifies whether to emit a `4xx` metric with the `CanaryName` dimension for this canary.

with5xxMetric(_5xxMetric)

Accepts a Boolean argument, which specifies whether to emit a `5xx` metric with the `CanaryName` dimension for this canary.

withAggregated2xxMetric(aggregated2xxMetric)

Accepts a Boolean argument, which specifies whether to emit a `2xx` metric with no dimension for this canary.

withAggregated4xxMetric(aggregated4xxMetric)

Accepts a Boolean argument, which specifies whether to emit a `4xx` metric with no dimension for this canary.

withAggregated5xxMetric(aggregated5xxMetric)

Accepts a Boolean argument, which specifies whether to emit a `5xx` metric with no dimension for this canary.

withAggregatedFailedCanaryMetric(aggregatedFailedCanaryMetric)

Accepts a Boolean argument, which specifies whether to emit a `Failed` metric with no dimension for this canary.

withAggregatedFailedRequestsMetric(aggregatedFailedRequestsMetric)

Accepts a Boolean argument, which specifies whether to emit a `Failed requests` metric with no dimension for this canary.

withFailedCanaryMetric(failedCanaryMetric)

Accepts a Boolean argument, which specifies whether to emit a `Failed` metric with the `CanaryName` dimension for this canary.

withFailedRequestsMetric(failedRequestsMetric)

Accepts a Boolean argument, which specifies whether to emit a `Failed requests` metric with the `CanaryName` dimension for this canary.

withStepDurationMetric(stepDurationMetric)

Accepts a Boolean argument, which specifies whether to emit a `Duration` metric with the `CanaryName` dimension for this canary.

withStepSuccessMetric(stepSuccessMetric)

Accepts a Boolean argument, which specifies whether to emit a `StepSuccess` metric with the `CanaryName` dimension for this canary.

Methods to enable or disable other features

withHarFile()

Accepts a Boolean argument, which specifies whether to create a HAR file for this canary.

withStepsReport()

Accepts a Boolean argument, which specifies whether to report a step execution summary for this canary.

withIncludeUrlPassword()

Accepts a Boolean argument, which specifies whether to include passwords that appear in URLs in logs and reports.

withRestrictedUrlParameters()

Accepts an array of URL path or query parameters to redact. This applies to URLs appearing in logs, reports, and errors. You can pass an asterisk (*) as a value to redact all URL path and query parameter values

withLogRequest()

Accepts a Boolean argument, which specifies whether to log every request in the canary's logs.

withLogResponse()

Accepts a Boolean argument, which specifies whether to log every response in the canary's logs.

withLogRequestBody()

Accepts a Boolean argument, which specifies whether to log every request body in the canary's logs.

withLogResponseBody()

Accepts a Boolean argument, which specifies whether to log every response body in the canary's logs.

withLogRequestHeaders()

Accepts a Boolean argument, which specifies whether to log every request header in the canary's logs.

withLogResponseHeaders()

Accepts a Boolean argument, which specifies whether to log every response header in the canary's logs.

getHarFile()

Returns whether the canary creates a HAR file.

getStepsReport()

Returns whether the canary reports a step execution summary.

getIncludeUrlPassword()

Returns whether the canary includes passwords that appear in URLs in logs and reports.

getRestrictedUrlParameters()

Returns whether the canary redacts URL path or query parameters.

getLogRequest()

Returns whether the canary logs every request in the canary's logs.

getLogResponse()

Returns whether the canary logs every response in the canary's logs.

getLogRequestBody()

Returns whether the canary logs every request body in the canary's logs.

getLogResponseBody()

Returns whether the canary logs every response body in the canary's logs.

getLogRequestHeaders()

Returns whether the canary logs every request header in the canary's logs.

getLogResponseHeaders()

Returns whether the canary logs every response header in the canary's logs.

Functions for all canaries

- `withIncludeRequestHeaders`(includeRequestHeaders)
- `withIncludeResponseHeaders`(includeResponseHeaders)
- `withRestrictedHeaders`(restrictedHeaders)
- `withIncludeRequestBody`(includeRequestBody)
- `withIncludeResponseBody`(includeResponseBody)
- `enableReportingOptions`()— Enables all reporting options-- **includeRequestHeaders**, **includeResponseHeaders**, **includeRequestBody**, and **includeResponseBody**, .
- `disableReportingOptions`()— Disables all reporting options-- **includeRequestHeaders**, **includeResponseHeaders**, **includeRequestBody**, and **includeResponseBody**, .

setConfig(options) for UI canaries

For UI canaries, **setConfig** can include the following Boolean parameters:

- `continueOnStepFailure` (boolean)— Whether to continue with running the canary script after a step fails (this refers to the **executeStep** function). If any steps fail, the canary run will still be marked as failed. The default is `false`.
- `harFile` (boolean)— Whether to create a HAR file. The default is `True`.
- `screenshotOnStepStart` (boolean)— Whether to take a screenshot before starting a step.
- `screenshotOnStepSuccess` (boolean)— Whether to take a screenshot after completing a successful step.
- `screenshotOnStepFailure` (boolean)— Whether to take a screenshot after a step fails.

Methods to enable or disable screenshots

disableStepScreenshots()

Disables all screenshot options (screenshotOnStepStart, screenshotOnStepSuccess, and screenshotOnStepFailure).

enableStepScreenshots()

Enables all screenshot options (screenshotOnStepStart, screenshotOnStepSuccess, and screenshotOnStepFailure). By default, all these methods are enabled.

getScreenshotOnStepFailure()

Returns whether the canary takes a screenshot after a step fails.

getScreenshotOnStepStart()

Returns whether the canary takes a screenshot before starting a step.

getScreenshotOnStepSuccess()

Returns whether the canary takes a screenshot after completing a step successfully.

withScreenshotOnStepStart(screenshotOnStepStart)

Accepts a Boolean argument, which indicates whether to take a screenshot before starting a step.

withScreenshotOnStepSuccess(screenshotOnStepSuccess)

Accepts a Boolean argument, which indicates whether to take a screenshot after completing a step successfully.

withScreenshotOnStepFailure(screenshotOnStepFailure)

Accepts a Boolean argument, which indicates whether to take a screenshot after a step fails.

Usage in UI canaries

First, import the synthetics dependency and fetch the configuration.

```
// Import Synthetics dependency
const synthetics = require('Synthetics');

// Get Synthetics configuration
const synConfig = synthetics.getConfiguration();
```

Then, set the configuration for each option by calling the setConfig method using one of the following options.

```
// Set configuration values
    synConfig.setConfig({
        screenshotOnStepStart: true,
        screenshotOnStepSuccess: false,
        screenshotOnStepFailure: false
    });
```

Or

```
synConfig.withScreenshotOnStepStart(false).withScreenshotOnStepSuccess(true).withScreenshotOnSt
```

To disable all screenshots, use the `disableStepScreenshots()` function as in this example.

```
synConfig.disableStepScreenshots();
```

You can enable and disable screenshots at any point in the code. For example, to disable screenshots only for one step, disable them before running that step and then enable them after the step.

setConfig(options) for API canaries

For API canaries, **setConfig** can include the following Boolean parameters:

- `continueOnHttpStepFailure` (boolean)— Whether to continue with running the canary script after an HTTP step fails (this refers to the **executeHttpStep** function). If any steps fail, the canary run will still be marked as failed. The default is `true`.

Visual monitoring

Visual monitoring compares screenshots taken during a canary run with screenshots taken during a baseline canary run. If the discrepancy between the two screenshots is beyond a threshold percentage, the canary fails and you can see the areas with differences highlighted in color in the canary run report. Visual monitoring is supported in canaries running **syn-puppeteer-node-3.2** and later. It is not currently supported in canaries running Python and Selenium.

To enable visual monitoring, add the following line of code to the canary script. For more details, see [SyntheticsConfiguration class](#).

```
syntheticsConfiguration.withVisualCompareWithBaseRun(true);
```

The first time that the canary runs successfully after this line is added to the script, it uses the screenshots taken during that run as the baseline for comparison. After that first canary run, you can use the CloudWatch console to edit the canary to do any of the following:

- Set the next run of the canary as the new baseline.
- Draw boundaries on the current baseline screenshot to designate areas of the screenshot to ignore during visual comparisons.
- Remove a screenshot from being used for visual monitoring.

For more information about using the CloudWatch console to edit a canary, see [Edit or delete a canary](#).

Other options for visual monitoring

syntheticsConfiguration.withVisualVarianceThresholdPercentage(desiredPercentage)

Set the acceptable percentage for screenshot variance in visual comparisons.

syntheticsConfiguration.withVisualVarianceHighlightHexColor("#fafa00")

Set the highlight color that designates variance areas when you look at canary run reports that use visual monitoring.

syntheticsConfiguration.withFailCanaryRunOnVisualVariance(failCanary)

Set whether or not the canary fails when there is a visual difference that is more than the threshold. The default is to fail the canary.

Synthetics logger

SyntheticsLogger writes logs out to both the console and to a local log file at the same log level. This log file is written to both locations only if the log level is at or below the desired logging level of the log function that was called.

The logging statements in the local log file are prepended with "DEBUG: ", "INFO: ", and so on to match the log level of the function that was called.

You can use the SyntheticsLogger, assuming you want to run the Synthetics Library at the same log level as your Synthetics canary logging.

Using the SyntheticsLogger is not required to create a log file that is uploaded to your S3 results location. You could instead create a different log file in the `/tmp` folder. Any files created under the `/tmp` folder are uploaded to the results location in S3 as artifacts.

To use the Synthetics Library logger:

```
const log = require('SyntheticsLogger');
```

Useful function definitions:

log.debug(*`message`*, *`ex`*);

Parameters: *`message`* is the message to log. *`ex`* is the exception, if any, to log.

Example:

```
log.debug("Starting step - login.");
```

log.error(*`message`*, *`ex`*);

Parameters: *`message`* is the message to log. *`ex`* is the exception, if any, to log.

Example:

```
try {
  await login();
catch (ex) {
  log.error("Error encountered in step - login.", ex);
}
```

log.info(*message*, *ex*);

Parameters: *message* is the message to log. *ex* is the exception, if any, to log.

Example:

```
log.info("Successfully completed step - login.");
```

log.log(*message*, *ex*);

This is an alias for `log.info`.

Parameters: *message* is the message to log. *ex* is the exception, if any, to log.

Example:

```
 log.log("Successfully completed step - login.");
```

log.warn(*message*, *ex*);

Parameters: *message* is the message to log. *ex* is the exception, if any, to log.

Example:

```
log.warn("Exception encountered trying to publish CloudWatch Metric.", ex);
```

SyntheticsLogHelper class

The `SyntheticsLogHelper` class is available in the runtime `syn-nodejs-puppeteer-3.2` and later runtimes. It is already initialized in the CloudWatch Synthetics library and is configured with Synthetics configuration. You can add this as a dependency in your script. This class enables you to sanitize URLs, headers, and error messages to redact sensitive information.

Note

Synthetics sanitizes all URLs and error messages it logs before including them in logs, reports, HAR files, and canary run errors based on the Synthetics configuration setting `restrictedUrlParameters`. You have to use `getSanitizedUrl` or `getSanitizedErrorMessage` only if you are logging URLs or errors in your script. Synthetics does not store any canary artifacts except for canary errors thrown by the script. Canary run artifacts are stored in your customer account. For more information, see Security considerations for Synthetics canaries.

Topics

- getSanitizedUrl(url, stepConfig = null)
- getSanitizedErrorMessage
- getSanitizedHeaders(headers, stepConfig=null)

getSanitizedUrl(url, stepConfig = null)

This function is available in `syn-nodejs-puppeteer-3.2` and later. It returns sanitized url strings based on the configuration. You can choose to redact sensitive URL parameters such as password and access_token by setting the property `restrictedUrlParameters`. By default, passwords in URLs are redacted. You can enable URL passwords if needed by setting `includeUrlPassword` to true.

This function throws an error if the URL passed in is not a valid URL.

Parameters

- *`url`* is a string and is the URL to sanitize.
- *`stepConfig`* (Optional) overrides the global Synthetics configuration for this function. If `stepConfig` is not passed in, the global configuration is used to sanitize the URL.

Example

This example uses the following sample URL: `https://example.com/learn/home?access_token=12345&token_type=Bearer&expires_in=1200`. In this example, `access_token` contains your sensitive information which shouldn't be logged. Note that the Synthetics services doesn't store any canary run artifacts. Artifacts such as logs, screenshots, and reports are all stored in an Amazon S3 bucket in your customer account.

The first step is to set the Synthetics configuration.

```
// Import Synthetics dependency
const synthetics = require('Synthetics');

// Import Synthetics logger for logging url
const log = require('SyntheticsLogger');

// Get Synthetics configuration
const synConfig = synthetics.getConfiguration();

// Set restricted parameters
synConfig.setConfig({
   restrictedUrlParameters: ['access_token'];
});
// Import SyntheticsLogHelper dependency
const syntheticsLogHelper = require('SyntheticsLogHelper');

const sanitizedUrl = syntheticsLogHelper.getSanitizedUrl('URL');



const urlConfig = {
   restrictedUrlParameters = ['*']
};
const sanitizedUrl = syntheticsLogHelper.getSanitizedUrl('URL', urlConfig);
logger.info('My example url is: ' + sanitizedUrl);
```

Next, sanitize and log the URL

```
// Import SyntheticsLogHelper dependency
const syntheticsLogHelper = require('SyntheticsLogHelper');

const sanitizedUrl = syntheticsLogHelper.getSanitizedUrl('https://example.com/learn/
home?access_token=12345&token_type=Bearer&expires_in=1200');
```

This logs the following in your canary log.

```
My example url is: https://example.com/learn/home?
access_token=REDACTED&token_type=Bearer&expires_in=1200
```

You can override the Synthetics configuration for a URL by passing in an optional parameter containing Synthetics configuration options, as in the following example.

```
const urlConfig = {
   restrictedUrlParameters = ['*']
};
const sanitizedUrl = syntheticsLogHelper.getSanitizedUrl('https://example.com/learn/
home?access_token=12345&token_type=Bearer&expires_in=1200', urlConfig);
logger.info('My example url is: ' + sanitizedUrl);
```

The preceding example redacts all query parameters, and is logged as follows:

```
My example url is: https://example.com/learn/home?
access_token=REDACTED&token_type=REDACTED&expires_in=REDACTED
```

getSanitizedErrorMessage

This function is available in `syn-nodejs-puppeteer-3.2` and later. It returns sanitized error strings by sanitizing any URLs present based on the Synthetics configuration. You can choose to override the global Synthetics configuration when you call this function by passing an optional `stepConfig` parameter.

Parameters

- *`error`* is the error to sanitize. It can be an Error object or a string.
- *`stepConfig`* (Optional) overrides the global Synthetics configuration for this function. If `stepConfig` is not passed in, the global configuration is used to sanitize the URL.

Example

This example uses the following error: `Failed to load url: https://example.com/learn/home?access_token=12345&token_type=Bearer&expires_in=1200`

The first step is to set the Synthetics configuration.

```
// Import Synthetics dependency
const synthetics = require('Synthetics');
```

```
// Import Synthetics logger for logging url
const log = require('SyntheticsLogger');

// Get Synthetics configuration
const synConfig = synthetics.getConfiguration();

// Set restricted parameters
synConfig.setConfig({
   restrictedUrlParameters: ['access_token'];
});
```

Next, sanitize and log the error message

```
// Import SyntheticsLogHelper dependency
const syntheticsLogHelper = require('SyntheticsLogHelper');

try {
   // Your code which can throw an error containing url which your script logs
} catch (error) {
    const sanitizedErrorMessage =
 syntheticsLogHelper.getSanitizedErrorMessage(errorMessage);
    logger.info(sanitizedErrorMessage);
}
```

This logs the following in your canary log.

```
Failed to load url: https://example.com/learn/home?
access_token=REDACTED&token_type=Bearer&expires_in=1200
```

getSanitizedHeaders(headers, stepConfig=null)

This function is available in `syn-nodejs-puppeteer-3.2` and later. It returns sanitized headers based on the `restrictedHeaders` property of `syntheticsConfiguration`. The headers specified in the `restrictedHeaders` property are redacted from logs, HAR files, and reports.

Parameters

- *`headers`* is an object containing the headers to sanitize.
- *`stepConfig`* (Optional) overrides the global Synthetics configuration for this function. If `stepConfig` is not passed in, the global configuration is used to sanitize the headers.

Node.js library classes and functions that apply to UI canaries only

The following CloudWatch Synthetics library functions for Node.js are useful only for UI canaries.

Topics

Synthetics class

The following functions are in the Synthetics class.

Topics

async addUserAgent(page, userAgentString);

This function appends *`userAgentString`* to the specified page's user-agent header.

Example:

```
await synthetics.addUserAgent(page, "MyApp-1.0");
```

Results in the page's user-agent header being set to *`browsers-user-agent-header-value`*`MyApp-1.0`

async executeStep(stepName, functionToExecute, [stepConfig]);

Executes the provided step, wrapping it with start/pass/fail logging, start/pass/fail screenshots, and pass/fail and duration metrics.

> **Note**
>
> If you are using the `syn-nodejs-2.1` or later runtime, you can configure whether and when screenshots are taken. For more information, see SyntheticsConfiguration class.

The `executeStep` function also does the following:

- Logs that the step started.
- Takes a screenshot named `<stepName>-starting`.
- Starts a timer.
- Executes the provided function.
- If the function returns normally, it counts as passing. If the function throws, it counts as failing.
- Ends the timer.
- Logs whether the step passed or failed
- Takes a screenshot named `<stepName>-succeeded` or `<stepName>-failed`.
- Emits the `stepName` `SuccessPercent` metric, 100 for pass or 0 for failure.
- Emits the `stepName` `Duration` metric, with a value based on the step start and end times.
- Finally, returns what the `functionToExecute` returned or re-throws what `functionToExecute` threw.

If the canary uses the `syn-nodejs-2.0` runtime or later, this function also adds a step execution summary to the canary's report. The summary includes details about each step, such as start time, end time, status (PASSED/FAILED), failure reason (if failed), and screenshots captured during the execution of each step.

Example:

```
await synthetics.executeStep('navigateToUrl', async function (timeoutInMillis = 30000)
 {
```

```
        await page.goto(url, {waitUntil: ['load', 'networkidle0'], timeout: 
 timeoutInMillis});});
```

Response:

Returns what `functionToExecute` returns.

Updates with syn-nodejs-2.2

Starting with `syn-nodejs-2.2`, you can optionally pass step configurations to override CloudWatch Synthetics configurations at the step level. For a list of options that you can pass to `executeStep`, see SyntheticsConfiguration class.

The following example overrides the default `false` configuration for `continueOnStepFailure` to `true` and specifies when to take screenthots.

```
var stepConfig = {
    'continueOnStepFailure': true,
    'screenshotOnStepStart': false,
    'screenshotOnStepSuccess': true,
    'screenshotOnStepFailure': false
}

await executeStep('Navigate to amazon', async function (timeoutInMillis = 30000) {
      await page.goto(url, {waitUntil: ['load', 'networkidle0'], timeout: 
 timeoutInMillis});
 }, stepConfig);
```

getDefaultLaunchOptions();

The `getDefaultLaunchOptions()` function returns the browser launch options that are used by CloudWatch Synthetics. For more information, see Launch options type

```
// This function returns default launch options used by Synthetics.
const defaultOptions = await synthetics.getDefaultLaunchOptions();
```

getPage();

Returns the current open page as a Puppeteer object. For more information, see Puppeteer API v1.14.0.

Example:

```
let page = await synthetics.getPage();
```

Response:

The page (Puppeteer object) that is currently open in the current browser session.

getRequestResponseLogHelper();

> **Important**
>
> In canaries that use the `syn-nodejs-puppeteer-3.2` runtime or later, this function is deprecated along with the `RequestResponseLogHelper` class. Any use of this function causes a warning to appear in your canary logs. This function will be removed in future runtime versions. If you are using this function, use [RequestResponseLogHelper class] instead.

Use this function as a builder pattern for tweaking the request and response logging flags.

Example:

```
synthetics.setRequestResponseLogHelper(getRequestResponseLogHelper().withLogRequestHeaders(fals
```

Response:

```
{RequestResponseLogHelper}
```

launch(options)

The options for this function are available only in the `syn-nodejs-2.1` runtime version or later.

This function is used only for UI canaries. It closes the existing browser and launches a new one.

> **Note**
>
> CloudWatch Synthetics always launches a browser before starting to run your script. You don't need to call launch() unless you want to launch a new browser with custom options.

(options) is a configurable set of options to set on the browser. For more information, Launch options type .

If you call this function with no options, Synthetics launches a browser with default arguments, `executablePath`, and `defaultViewport`. The default viewport in CloudWatch Synthetics is 1920 by 1080.

You can override launch parameters used by CloudWatch Synthetics and pass additional parameters when launching the browser. For example, the following code snippet launches a browser with default arguments and a default executable path, but with a viewport of 800 x 600.

```
await synthetics.launch({
        defaultViewport: {
            "deviceScaleFactor": 1,
            "width": 800,
            "height": 600
    }});
```

The following sample code adds a new `ignoreHTTPSErrors` parameter to the CloudWatch Synthetics launch parameters:

```
await synthetics.launch({
        ignoreHTTPSErrors: true
 });
```

You can disable web security by adding a `--disable-web-security` flag to args in the CloudWatch Synthetics launch parameters:

```
// This function adds the --disable-web-security flag to the launch parameters
const defaultOptions = await synthetics.getDefaultLaunchOptions();
const launchArgs = [...defaultOptions.args, '--disable-web-security'];
await synthetics.launch({
     args: launchArgs
  });
```

RequestResponseLogHelper class

Important

In canaries that use the `syn-nodejs-puppeteer-3.2` runtime or later, this class is deprecated. Any use of this class causes a warning to appear in your canary logs. This function will be removed in future runtime versions. If you are using this function, use [RequestResponseLogHelper class](#) instead.

Handles the fine-grained configuration and creation of string representations of request and response payloads.

```
class RequestResponseLogHelper {

    constructor () {
        this.request = {url: true, resourceType: false, method: false, headers: false,
 postData: false};
        this.response = {status: true, statusText: true, url: true, remoteAddress:
 false, headers: false};
    }

    withLogRequestUrl(logRequestUrl);

    withLogRequestResourceType(logRequestResourceType);

    withLogRequestMethod(logRequestMethod);

    withLogRequestHeaders(logRequestHeaders);

    withLogRequestPostData(logRequestPostData);


    withLogResponseStatus(logResponseStatus);

    withLogResponseStatusText(logResponseStatusText);

    withLogResponseUrl(logResponseUrl);

    withLogResponseRemoteAddress(logResponseRemoteAddress);

    withLogResponseHeaders(logResponseHeaders);
```

Example:

```
synthetics.setRequestResponseLogHelper(getRequestResponseLogHelper()
.withLogRequestPostData(true)
.withLogRequestHeaders(true)
```

```
.withLogResponseHeaders(true));
```

Response:

```
{RequestResponseLogHelper}
```

setRequestResponseLogHelper();

> **Important**
>
> In canaries that use the `syn-nodejs-puppeteer-3.2` runtime or later, this function is deprecated along with the `RequestResponseLogHelper` class. Any use of this function causes a warning to appear in your canary logs. This function will be removed in future runtime versions. If you are using this function, use RequestResponseLogHelper class instead.

Use this function as a builder pattern for setting the request and response logging flags.

Example:

```
synthetics.setRequestResponseLogHelper().withLogRequestHeaders(true).withLogResponseHeaders(tru
```

Response:

```
{RequestResponseLogHelper}
```

async takeScreenshot(name, suffix);

Takes a screenshot (.PNG) of the current page with name and suffix (optional).

Example:

```
await synthetics.takeScreenshot("navigateToUrl", "loaded")
```

This example captures and uploads a screenshot named `01-navigateToUrl-loaded.png` to the canary's S3 bucket.

You can take a screenshot for a particular canary step by passing the `stepName` as the first parameter. Screenshots are linked to the canary step in your reports, to help you track each step while debugging.

CloudWatch Synthetics canaries automatically take screenshots before starting a step (the `executeStep` function) and after the step completion (unless you configure the canary to disable screenshots). You can take more screenshots by passing in the step name in the `takeScreenshot` function.

The following example takes screenshot with the `signupForm` as the value of the `stepName`. The screenshot will be named `02-signupForm-address` and will be linked to the step named `signupForm` in the canary report.

```
await synthetics.takeScreenshot('signupForm', 'address')
```

BrokenLinkCheckerReport class

This class provides methods to add a synthetics link. It's supported only on canaries that use the `syn-nodejs-2.0-beta` version of the runtime or later.

To use `BrokenLinkCheckerReport`, include the following lines in the script:

```
const BrokenLinkCheckerReport = require('BrokenLinkCheckerReport');

const brokenLinkCheckerReport = new BrokenLinkCheckerReport();
```

Useful function definitions:

addLink(*syntheticsLink*, isBroken)

`syntheticsLink` is a `SyntheticsLink` object representing a link. This function adds the link according to the status code. By default, it considers a link to be broken if the status code is not available or the status code is 400 or higher. You can override this default behavior by passing in the optional parameter `isBrokenLink` with a value of `true` or `false`.

This function does not have a return value.

getLinks()

This function returns an array of `SyntheticsLink` objects that are included in the broken link checker report.

getTotalBrokenLinks()

This function returns a number representing the total number of broken links.

getTotalLinksChecked()

This function returns a number representing the total number of links included in the report.

How to use BrokenLinkCheckerReport

The following canary script code snippet demonstrates an example of navigating to a link and adding it to the broken link checker report.

1. Import `SyntheticsLink`, `BrokenLinkCheckerReport`, and `Synthetics`.

```
const BrokenLinkCheckerReport = require('BrokenLinkCheckerReport');
const SyntheticsLink = require('SyntheticsLink');

// Synthetics dependency
const synthetics = require('Synthetics');
```

2. To add a link to the report, create an instance of `BrokenLinkCheckerReport`.

```
let brokenLinkCheckerReport = new BrokenLinkCheckerReport();
```

3. Navigate to the URL and add it to the broken link checker report.

```
let url = "https://amazon.com";

let syntheticsLink = new SyntheticsLink(url);

// Navigate to the url.
let page = await synthetics.getPage();

// Create a new instance of Synthetics Link
let link = new SyntheticsLink(url)

try {
    const response = await page.goto(url, {waitUntil: 'domcontentloaded', timeout:
 30000});
} catch (ex) {
    // Add failure reason if navigation fails.
    link.withFailureReason(ex);
```

```
}

if (response) {
    // Capture screenshot of destination page
    let screenshotResult = await synthetics.takeScreenshot('amazon-home', 'loaded');

    // Add screenshot result to synthetics link
    link.addScreenshotResult(screenshotResult);

    // Add status code and status description to the link
    link.withStatusCode(response.status()).withStatusText(response.statusText())
}

// Add link to broken link checker report.
brokenLinkCheckerReport.addLink(link);
```

4. Add the report to Synthetics. This creates a JSON file named `BrokenLinkCheckerReport.json` in your S3 bucket for each canary run. You can see a links report in the console for each canary run along with screenshots, logs, and HAR files.

```
await synthetics.addReport(brokenLinkCheckerReport);
```

SyntheticsLink class

This class provides methods to wrap information. It's supported only on canaries that use the `syn-nodejs-2.0-beta` version of the runtime or later.

To use `SyntheticsLink`, include the following lines in the script:

```
const SyntheticsLink = require('SyntheticsLink');

const syntheticsLink = new SyntheticsLink("https://www.amazon.com");
```

This function returns `syntheticsLink`*`Object`*

Useful function definitions:

withUrl(*`url`*)

`url` is a URL string. This function returns `syntheticsLink`*`Object`*

withText(*`text`*)

text is a string representing anchor text. This function returns syntheticsLink*Object*. It adds anchor text corresponding to the link.

withParentUrl(*parentUrl*)

parentUrl is a string representing the parent (source page) URL. This function returns syntheticsLink*Object*

withStatusCode(*statusCode*)

statusCode is a string representing the status code. This function returns syntheticsLink*Object*

withFailureReason(*failureReason*)

failureReason is a string representing the failure reason. This function returns syntheticsLink*Object*

addScreenshotResult(*screenshotResult*)

screenshotResult is an object. It is an instance of ScreenshotResult that was returned by the Synthetics function takeScreenshot. The object includes the following:

- fileName— A string representing the screenshotFileName
- pageUrl (optional)
- error (optional)

Node.js library classes and functions that apply to API canaries only

The following CloudWatch Synthetics library functions for Node.js are useful only for API canaries.

Topics

- executeHttpStep(stepName, requestOptions, [callback], [stepConfig])

executeHttpStep(stepName, requestOptions, [callback], [stepConfig])

Executes the provided HTTP request as a step, and publishes SuccessPercent (pass/fail) and Duration metrics.

executeHttpStep uses either HTTP or HTTPS native functions under the hood, depending upon the protocol specified in the request.

This function also adds a step execution summary to the canary's report. The summary includes details about each HTTP request, such as the following:

- Start time
- End time
- Status (PASSED/FAILED)
- Failure reason, if it failed
- HTTP call details such as request/response headers, body, status code, status message, and performance timings.

Topics

Parameters

stepName(*String*)

Specifies the name of the step. This name is also used for publishing CloudWatch metrics for this step.

requestOptions(*Object or String*)

The value of this parameter can be a URL, a URL string, or an object. If it is an object, then it must be a set of configurable options to make an HTTP request. It supports all options in http.request(options[, callback]) in the Node.js documentation.

In addition to these Node.js options, **requestOptions** supports the additional parameter body. You can use the body parameter to pass data as a request body.

callback(*response*)

(Optional) This is a user function which is invoked with the HTTP response. The response is of the type Class: http.IncomingMessage.

stepConfig(*object*)

(Optional) Use this parameter to override global synthetics configurations with a different configuration for this step.

Examples of using executeHttpStep

The following series of examples build on each other to illustrate the various uses of this option.

This first example configures request parameters. You can pass a URL as **requestOptions**:

```
let requestOptions = 'https://www.amazon.com';
```

Or you can pass a set of options:

```
let requestOptions = {
        'hostname': 'myproductsEndpoint.com',
        'method': 'GET',
        'path': '/test/product/validProductName',
        'port': 443,
        'protocol': 'https:'
    };
```

The next example creates a callback function which accepts a response. By default, if you do not specify **callback**, CloudWatch Synthetics validates that the status is between 200 and 299 inclusive.

```
// Handle validation for positive scenario
    const callback = async function(res) {
        return new Promise((resolve, reject) => {
            if (res.statusCode < 200 || res.statusCode > 299) {
                throw res.statusCode + ' ' + res.statusMessage;
            }

            let responseBody = '';
            res.on('data', (d) => {
                responseBody += d;
            });

            res.on('end', () => {
                // Add validation on 'responseBody' here if required. For ex, your
 status code is 200 but data might be empty
                resolve();
            });
        });
    };
```

The next example creates a configuration for this step that overrides the global CloudWatch Synthetics configuration. The step configuration in this example allows request headers, response headers, request body (post data), and response body in your report and restrict 'X-Amz-Security-Token' and 'Authorization' header values. By default, these values are not included in the report for security reasons. If you choose to include them, the data is only stored in your S3 bucket.

```
// By default headers, post data, and response body are not included in the report for
 security reasons.
// Change the configuration at global level or add as step configuration for individual
 steps
let stepConfig = {
    includeRequestHeaders: true,
    includeResponseHeaders: true,
    restrictedHeaders: ['X-Amz-Security-Token', 'Authorization'], // Restricted header
 values do not appear in report generated.
    includeRequestBody: true,
    includeResponseBody: true
};
```

This final example passes your request to **executeHttpStep** and names the step.

```
await synthetics.executeHttpStep('Verify GET products API', requestOptions, callback,
 stepConfig);
```

With this set of examples, CloudWatch Synthetics adds the details from each step in your report and produces metrics for each step using **stepName**.

You will see `successPercent` and `duration` metrics for the `Verify GET products API` step. You can monitor your API performance by monitoring the metrics for your API call steps.

For a sample complete script that uses these functions, see Multi-step API canary.

Library functions available for Python canary scripts using Selenium

This section lists the Selenium library functions available for Python canary scripts.

Topics

- Python and Selenium library classes and functions that apply to all canaries
- Python and Selenium library classes and functions that apply to UI canaries only

Python and Selenium library classes and functions that apply to all canaries

The following CloudWatch Synthetics Selenium library functions for Python are useful for all canaries.

Topics

- SyntheticsConfiguration class
- SyntheticsLogger class

SyntheticsConfiguration class

You can use the SyntheticsConfiguration class to configure the behavior of Synthetics library functions. For example, you can use this class to configure the `executeStep()` function to not capture screenshots.

You can set CloudWatch Synthetics configurations at the global level.

Function definitions:

set_config(options)

```
from aws_synthetics.common import synthetics_configuration
```

`options` is an object, which is a set of configurable options for your canary. The following sections explain the possible fields in *`options`*.

- `screenshot_on_step_start` (boolean)— Whether to take a screenshot before starting a step.
- `screenshot_on_step_success` (boolean)— Whether to take a screenshot after completing a successful step.
- `screenshot_on_step_failure` (boolean)— Whether to take a screenshot after a step fails.

with_screenshot_on_step_start(screenshot_on_step_start)

Accepts a Boolean argument, which indicates whether to take a screenshot before starting a step.

with_screenshot_on_step_success(screenshot_on_step_success)

Accepts a Boolean argument, which indicates whether to take a screenshot after completing a step successfully.

with_screenshot_on_step_failure(screenshot_on_step_failure)

Accepts a Boolean argument, which indicates whether to take a screenshot after a step fails.

get_screenshot_on_step_start()

Returns whether to take a screenshot before starting a step.

get_screenshot_on_step_success()

Returns whether to take a screenshot after completing a step successfully.

get_screenshot_on_step_failure()

Returns whether to take a screenshot after a step fails.

disable_step_screenshots()

Disables all screenshot options (get_screenshot_on_step_start, get_screenshot_on_step_success, and get_screenshot_on_step_failure).

enable_step_screenshots()

Enables all screenshot options (get_screenshot_on_step_start, get_screenshot_on_step_success, and get_screenshot_on_step_failure). By default, all these methods are enabled.

setConfig(options) regarding CloudWatch metrics

For canaries using `syn-python-selenium-1.1` or later, the **(options)** for **setConfig** can include the following Boolean parameters that determine which metrics are published by the canary. The default for each of these options is `true`. The options that start with `aggregated` determine whether the metric is emitted without the `CanaryName` dimension. You can use these metrics to see the aggregated results for all of your canaries. The other options determine whether the metric is emitted with the `CanaryName` dimension. You can use these metrics to see results for each individual canary.

For a list of CloudWatch metrics emitted by canaries, see CloudWatch metrics published by canaries.

- `failed_canary_metric` (boolean)— Whether to emit the `Failed` metric (with the `CanaryName` dimension) for this canary. The default is `true`.
- `failed_requests_metric` (boolean)— Whether to emit the `Failed requests` metric (with the `CanaryName` dimension) for this canary. The default is `true`.

- `2xx_metric` (boolean)— Whether to emit the `2xx` metric (with the `CanaryName` dimension) for this canary. The default is `true`.
- `4xx_metric` (boolean)— Whether to emit the `4xx` metric (with the `CanaryName` dimension) for this canary. The default is `true`.
- `5xx_metric` (boolean)— Whether to emit the `5xx` metric (with the `CanaryName` dimension) for this canary. The default is `true`.
- `step_duration_metric` (boolean)— Whether to emit the `Step duration` metric (with the `CanaryName` `StepName` dimensions) for this canary. The default is `true`.
- `step_success_metric` (boolean)— Whether to emit the `Step success` metric (with the `CanaryName` `StepName` dimensions) for this canary. The default is `true`.
- `aggregated_failed_canary_metric` (boolean)— Whether to emit the `Failed` metric (without the `CanaryName` dimension) for this canary. The default is `true`.
- `aggregated_failed_requests_metric` (boolean)— Whether to emit the `Failed Requests` metric (without the `CanaryName` dimension) for this canary. The default is `true`.
- `aggregated_2xx_metric` (boolean)— Whether to emit the `2xx` metric (without the `CanaryName` dimension) for this canary. The default is `true`.
- `aggregated_4xx_metric` (boolean)— Whether to emit the `4xx` metric (without the `CanaryName` dimension) for this canary. The default is `true`.
- `aggregated_5xx_metric` (boolean)— Whether to emit the `5xx` metric (without the `CanaryName` dimension) for this canary. The default is `true`.

with_2xx_metric(2xx_metric)

Accepts a Boolean argument, which specifies whether to emit a `2xx` metric with the `CanaryName` dimension for this canary.

with_4xx_metric(4xx_metric)

Accepts a Boolean argument, which specifies whether to emit a `4xx` metric with the `CanaryName` dimension for this canary.

with_5xx_metric(5xx_metric)

Accepts a Boolean argument, which specifies whether to emit a `5xx` metric with the `CanaryName` dimension for this canary.

withAggregated2xxMetric(aggregated2xxMetric)

Accepts a Boolean argument, which specifies whether to emit a `2xx` metric with no dimension for this canary.

withAggregated4xxMetric(aggregated4xxMetric)

Accepts a Boolean argument, which specifies whether to emit a `4xx` metric with no dimension for this canary.

with_aggregated_5xx_metric(aggregated_5xx_metric)

Accepts a Boolean argument, which specifies whether to emit a `5xx` metric with no dimension for this canary.

with_aggregated_failed_canary_metric(aggregated_failed_canary_metric)

Accepts a Boolean argument, which specifies whether to emit a `Failed` metric with no dimension for this canary.

with_aggregated_failed_requests_metric(aggregated_failed_requests_metric)

Accepts a Boolean argument, which specifies whether to emit a `Failed requests` metric with no dimension for this canary.

with_failed_canary_metric(failed_canary_metric)

Accepts a Boolean argument, which specifies whether to emit a `Failed` metric with the `CanaryName` dimension for this canary.

with_failed_requests_metric(failed_requests_metric)

Accepts a Boolean argument, which specifies whether to emit a `Failed requests` metric with the `CanaryName` dimension for this canary.

with_step_duration_metric(step_duration_metric)

Accepts a Boolean argument, which specifies whether to emit a `Duration` metric with the `CanaryName` dimension for this canary.

with_step_success_metric(step_success_metric)

Accepts a Boolean argument, which specifies whether to emit a `StepSuccess` metric with the `CanaryName` dimension for this canary.

Methods to enable or disable metrics

disable_aggregated_request_metrics()

Disables the canary from emitting all request metrics that are emitted with no `CanaryName` dimension.

disable_request_metrics()

Disables all request metrics, including both per-canary metrics and metrics aggregated across all canaries.

disable_step_metrics()

Disables all step metrics, including both step success metrics and step duration metrics.

enable_aggregated_request_metrics()

Enables the canary to emit all request metrics that are emitted with no `CanaryName` dimension.

enable_request_metrics()

Enables all request metrics, including both per-canary metrics and metrics aggregated across all canaries.

enable_step_metrics()

Enables all step metrics, including both step success metrics and step duration metrics.

Usage in UI canaries

First, import the synthetics dependency and fetch the configuration. Then, set the configuration for each option by calling the setConfig method using one of the following options.

```
from aws_synthetics.common import synthetics_configuration

synthetics_configuration.set_config(
     {
        "screenshot_on_step_start": False,
        "screenshot_on_step_success": False,
        "screenshot_on_step_failure": True
     }
)
```

```
or
```

Or

```
synthetics_configuration.with_screenshot_on_step_start(False).with_screenshot_on_step_success(F
```

To disable all screenshots, use the disableStepScreenshots() function as in this example.

```
synthetics_configuration.disable_step_screenshots()
```

You can enable and disable screenshots at any point in the code. For example, to disable screenshots only for one step, disable them before running that step and then enable them after the step.

set_config(options) for UI canaries

Starting with `syn-python-selenium-1.1`, for UI canaries, `set_config` can include the following Boolean parameters:

- `continue_on_step_failure` (boolean)— Whether to continue with running the canary script after a step fails (this refers to the **executeStep** function). If any steps fail, the canary run will still be marked as failed. The default is `false`.

SyntheticsLogger class

`synthetics_logger` writes logs out to both the console and to a local log file at the same log level. This log file is written to both locations only if the log level is at or below the desired logging level of the log function that was called.

The logging statements in the local log file are prepended with "DEBUG: ", "INFO: ", and so on to match the log level of the function that was called.

Using `synthetics_logger` is not required to create a log file that is uploaded to your Amazon S3 results location. You could instead create a different log file in the `/tmp` folder. Any files created under the `/tmp` folder are uploaded to the results location in the S3 bucket as artifacts.

To use `synthetics_logger`:

```
from aws_synthetics.common import synthetics_logger
```

Useful function definitions:

Get log level:

```
log_level = synthetics_logger.get_level()
```

Set log level:

```
synthetics_logger.set_level()
```

Log a message with a specified level. The level can be `DEBUG`, `INFO`, `WARN`, or `ERROR`, as in the following syntax examples:

```
synthetics_logger.debug(message, *args, **kwargs)
```

```
synthetics_logger.info(message, *args, **kwargs)
```

```
synthetics_logger.log(message, *args, **kwargs)
```

```
synthetics_logger.warn(message, *args, **kwargs)
```

```
synthetics_logger.error(message, *args, **kwargs)
```

For information about debug parameters, see the standard Python documentation at logging.debug

In these logging functions, the `message` is the message format string. The `args` are the arguments that are merged into `msg` using the string formatting operator.

There are three keyword arguments in `kwargs`:

- `exc_info`– If not evaluated as false, adds exception information to the logging message.
- `stack_info`– defaults to false. If true, adds stack information to the logging message, including the actual logging call.
- `extra`– The third optional keyword argument, which you can use to pass in a dictionary that is used to populate the `__dict__` of the `LogRecord` created for the logging event with user-defined attributes.

Examples:

Log a message with the level DEBUG:

```
synthetics_logger.debug('Starting step - login.')
```

Log a message with the level INFO. `logger.log` is a synonym for `logger.info`:

```
synthetics_logger.info('Successfully completed step - login.')
```

or

```
synthetics_logger.log('Successfully completed step - login.')
```

Log a message with the level WARN:

```
synthetics_logger.warn('Warning encountered trying to publish %s', 'CloudWatch Metric')
```

Log a message with the level ERROR:

```
synthetics_logger.error('Error encountered trying to publish %s', 'CloudWatch Metric')
```

Log an exception:

```
synthetics_logger.exception(message, *args, **kwargs)
```

Logs a message with level ERROR. Exception information is added to the logging message. You should call this function only from an exception handler.

For information about exception parameters, see the standard Python documentation at logging.exception

The `message` is the message format string. The `args` are the arguments, which are merged into `msg` using the string formatting operator.

There are three keyword arguments in `kwargs`:

- `exc_info`– If not evaluated as false, adds exception information to the logging message.
- `stack_info`– defaults to false. If true, adds stack information to the logging message, including the actual logging call.

- `extra`– The third optional keyword argument, which you can use to pass in a dictionary that is used to populate the `__dict__` of the `LogRecord` created for the logging event with user-defined attributes.

Example:

```
synthetics_logger.exception('Error encountered trying to publish %s', 'CloudWatch
 Metric')
```

Python and Selenium library classes and functions that apply to UI canaries only

The following CloudWatch Synthetics Selenium library functions for Python are useful only for UI canaries.

Topics

- SyntheticsBrowser class
- SyntheticsWebDriver class

SyntheticsBrowser class

When you create a browser instance by calling `synthetics_webdriver.Chrome()`, the returned browser instance is of the type `SyntheticsBrowser`. The `SyntheticsBrowser` class controls the ChromeDriver, and enables the canary script to drive the browser, allowing the Selenium WebDriver to work with Synthetics.

In addition to the standard Selenium methods, it also provides the following methods.

Topics

- set_viewport_size(width, height)
- save_screenshot(filename, suffix)

set_viewport_size(width, height)

Sets the viewport of the browser. Example:

```
browser.set_viewport_size(1920, 1080)
```

save_screenshot(filename, suffix)

Saves screenshots to the `/tmp` directory. The screenshots are uploaded from there to the canary artifacts folder in the S3 bucket.

filename is the file name for the screenshot, and *suffix* is an optional string to be used for naming the screenshot.

Example:

```
browser.save_screenshot('loaded.png', 'page1')
```

SyntheticsWebDriver class

To use this class, use the following in your script:

```
from aws_synthetics.selenium import synthetics_webdriver
```

Topics

add_execution_error(errorMessage, ex);

`errorMessage` describes the error and `ex` is the exception that is encountered

You can use `add_execution_error` to set execution errors for your canary. It fails the canary without interrupting the script execution. It also doesn't impact your `successPercent` metrics.

You should track errors as execution errors only if they are not important to indicate the success or failure of your canary script.

An example of the use of `add_execution_error` is the following. You are monitoring the availability of your endpoint and taking screenshots after the page has loaded. Because the failure of taking a screenshot doesn't determine availability of the endpoint, you can catch any errors encountered while taking screenshots and add them as execution errors. Your availability metrics will still indicate that the endpoint is up and running, but your canary status will be marked as failed. The following sample code block catches such an error and adds it as an execution error.

```
try:
    browser.save_screenshot("loaded.png")
except Exception as ex:
   self.add_execution_error("Unable to take screenshot", ex)
```

add_user_agent(user_agent_str)

Appends the value of `user_agent_str` to the browser's user agent header. You must assign `user_agent_str` before creating the browser instance.

Example:

```
synthetics_webdriver.add_user_agent('MyApp-1.0')
```

execute_step(step_name, function_to_execute)

Processes one function. It also does the following:

- Logs that the step started.
- Takes a screenshot named `<stepName>-starting`.
- Starts a timer.
- Executes the provided function.
- If the function returns normally, it counts as passing. If the function throws, it counts as failing.
- Ends the timer.
- Logs whether the step passed or failed
- Takes a screenshot named `<stepName>-succeeded` or `<stepName>-failed`.
- Emits the `stepName` `SuccessPercent` metric, 100 for pass or 0 for failure.
- Emits the `stepName` `Duration` metric, with a value based on the step start and end times.
- Finally, returns what the `functionToExecute` returned or re-throws what `functionToExecute` threw.

Example:

```
from selenium.webdriver.common.by import By
```

```
def custom_actions():
        #verify contains
        browser.find_element(By.XPATH, "//*[@id=\"id_1\"][contains(text(),'login')]")
        #click a button
        browser.find_element(By.XPATH, '//*[@id="submit"]/a').click()

    await synthetics_webdriver.execute_step("verify_click", custom_actions)
```

get_http_response(url)

Makes an HTTP request to the provided URL and returns the response code of the HTTP request. If an exception occurred during the HTTP request, a string with value "error" is returned instead.

Example:

```
response_code = syn_webdriver.get_http_response(url)
if not response_code or response_code == "error" or response_code < 200 or
 response_code > 299:
    raise Exception("Failed to load page!")
```

Chrome()

Launches an instance of the Chromium browser and returns the created instance of the browser.

Example:

```
browser = synthetics_webdriver.Chrome()
browser.get("https://example.com/)
```

To launch a browser in incognito mode, use the following:

```
add_argument('—incognito')
```

To add proxy settings, use the following:

```
add_argument('--proxy-server=%s' % PROXY)
```

Example:

```
from selenium.webdriver.chrome.options import Options
chrome_options = Options()
```

```
chrome_options.add_argument("——incognito")
browser = syn_webdriver.Chrome(chrome_options=chrome_options)
```

Scheduling canary runs using cron

Using a cron expression gives you flexibility when you schedule a canary. Cron expressions contain five or six fields in the order listed in the following table. The fields are separated by spaces. The syntax differs depending on whether you are using the CloudWatch console to create the canary, or the Amazon CLI or Amazon SDKs. When you use the console, you specify only the first five fields. When you use the Amazon CLI or Amazon SDKs, you specify all six fields, and you must specify * for the `Year` field.

Field	Allowed values	Allowed special characters
Minutes	0-59	, - * /
Hours	0-23	, - * /
Day-of-month	1-31	, - * ? / L W
Month	1-12 or JAN-DEC	, - * /
Day-of-week	1-7 or SUN-SAT	, - * ? L #
Year	*	

Special characters

- The **,** (comma) includes multiple values in the expression for a field. For example, in the Month field, JAN,FEB,MAR would include January, February, and March.
- The **-** (dash) special character specifies ranges. In the Day field, 1-15 would include days 1 through 15 of the specified month.
- The ***** (asterisk) special character includes all values in the field. In the Hours field, ***** includes every hour. You cannot use ***** in both the Day-of-month and Day-of-week fields in the same expression. If you use it in one, you must use **?** in the other.
- The **/** (forward slash) specifies increments. In the Minutes field, you can enter 1/10 to specify every tenth minute, starting from the first minute of the hour (for example, the eleventh, twenty-first, and thirty-first minute, and so on).

- The **?** (question mark) specifies one or another. If you enter **7** in the Day-of-month field and you don't care what day of the week the seventh is, you can enter **?** in the Day-of-week field.
- The **L** wildcard in the Day-of-month or Day-of-week fields specifies the last day of the month or week.
- The **W** wildcard in the Day-of-month field specifies a weekday. In the Day-of-month field, **3W** specifies the weekday closest to the third day of the month.
- The **#** wildcard in the Day-of-week field specifies a certain instance of the specified day of the week within a month. For example, 3#2 is the second Tuesday of the month. The 3 refers to Tuesday because it is the third day of each week, and the 2 refers to the second day of that type within the month.

Limitations

- You can't specify the Day-of-month and Day-of-week fields in the same cron expression. If you specify a value or `*` (asterisk) in one of the fields, you must use a **?** (question mark) in the other.
- Cron expressions that lead to rates faster than one minute are not supported.
- You can't set a canary to wait for more than a year before running, so you can specify only `*` in the `Year` field.

Examples

You can refer to the following sample cron strings when you create a canary. The following examples are the correct syntax for using the Amazon CLI or Amazon SDKs to create or update a canary. If you are using the CloudWatch console, omit the final `*` in each example.

Expression	Meaning
`0 10 * * ? *`	Run at 10:00 am (UTC) every day
`15 12 * * ? *`	Run at 12:15 pm (UTC) every day
`0 18 ? * MON-FRI *`	Run at 6:00 pm (UTC) every Monday through Friday
`0 8 1 * ? *`	Run at 8:00 am (UTC) on the first day of each month
`0/10 * ? * MON-SAT *`	Run every 10 minutes Monday through Saturday of each week

Expression	Meaning
`0/5 8-17 ? * MON-FRI *`	Run every five minutes Monday through Friday between 8:00 am and 5:55 pm (UTC)

Configuring your canary to retry automatically

When you create or update a canary, you can configure your canaries to automatically attempt additional runs when the scheduled one fails. This helps differentiate between genuine failures and temporary glitches, providing more reliable results. This feature is ideal for building more resilient monitoring systems while reducing false alarms and manual intervention.

To create a auto retry canary

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Application Signals**, **Synthetics Canaries**.
3. Choose **Create Canary**.
4. Under **Additional configuration**, **Auto-retry,** select the desired maximum retry number.

To update the maximum retry number for a canary

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Application Signals**, **Synthetics Canaries**.
3. You can do one of the following:
   - Select the canary and choose **Actions**, **Enable auto-retry**, and adjust the maximum retries.
   - Select the canary and choose **Actions**, **Edit**. In the **Edit detail** page, under **Additional configuration**, **Auto-retry**, adjust the retry configuration.

Limitations

Here are the limitations to configure auto retry.

- Supported only on runtime versions `syn-nodejs-puppeteer-10.0` or newer, `syn-nodejs-playwright-2.0` or newer, or `syn-python-selenium-5.1` or newer

- Long running canaries which timeout after ten minutes are limited to one retry. All other canaries can support upto two retries

Groups

You can create *groups* to associate canaries with each other, including cross-Region canaries. Using groups can help you with managing and automating your canaries, and you can also view aggregated run results and statistics for all canaries in a group.

Groups are global resources. When you create a group, it is replicated across all Amazon Regions that support groups, and you can add canaries from any of these Regions to it, and view it in any of these Regions. Although the group ARN format reflects the Region name where it was created, a group is not constrained to any Region. This means that you can put canaries from multiple Regions into the same group, and then use that group to view and manage all of those canaries in a single view.

Groups are supported in all Regions except the Regions that are disabled by default. For more information about these Regions, see Enabling a Region .

Each group can contain as many as 10 canaries. You can have as many as 20 groups in your account. Any single canary can be a member of up to 10 groups.

To create a group

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Application Signals**, **Synthetics Canaries**.
3. Choose **Create Group**.
4. Under **Group Name**, enter a name for the group.
5. Select canaries to associate with this group. To select a canary, type its complete name in **Exact canary name** and choose **Search**. Then select the check box next to the canary name. If there are multiple canaries with the same name in different Regions, be sure to select the canaries that you want.

   You can repeat this step to associate as many as 10 canaries with the group.
6. (Optional) Under **Tags**, add one or more key-value pairs as tags for this group. Tags can help you identify and organize your Amazon resources and track your Amazon costs. For more information, see Tagging your Amazon CloudWatch resources.

7. Choose **Create Group**.

Test a canary locally

This section explains how to modify, test, and debug CloudWatch Synthetics canaries directly within the Microsoft Visual Studio code editor or the JetBrains IDE code editor. The local debugging environment uses a Serverless Application Model (SAM) container to simulate a Lambda function to emulate the behavior of a Synthetics canary.

> **Note**
>
> It is impractical to perform locally debug canaries that rely on visual monitoring. Visual monitoring rely on capturing base screenshots during an initial run, and then comparing these screenshots to the screenshots from subsequent runs. In a local development environment, runs are not stored or tracked, and each iteration is an independent, standalone run. The absence of a canary run history makes it impractical to debug canaries that rely on visual monitoring.

Prerequisites

1. Choose or create an Amazon S3 bucket to use to store artifacts from local canary test runs, such as HAR files and screenshots. This requires you to be provisioned with IAM. If you skip setting up Amazon S3 buckets you can still test your canary locally, but you will see an error message about the missing bucket and you won't have access to canary artifacts.

   If you use an Amazon S3 bucket, we recommend that you set the bucket lifecycle to delete objects after a few days, to save costs. For more information, see Managing your storage lifecycle.
2. Set up a default Amazon profile for your Amazon account. For more information, see Configuration and credential file settings.
3. Set the debug environment's default Amazon Region to your preferred Region, such as `us-west-2`.
4. Install the Amazon SAM CLI. For more information, see Installing the Amazon SAM CLI.
5. Install Visual Studio Code Editor or JetBrains IDE. For more information, see Visual Studio Code or JetBrains IDE

6. Install Docker to work with the Amazon SAM CLI. Make sure to start the docker daemon. For more information, see Installing Docker to use with the Amazon SAM CLI.

   Alternatively, You can install other container management software such as Rancher, as long as it uses the Docker runtime.

7. Install an Amazon toolkit extension for your preferred editor. For more information, see Installing the Amazon Toolkit for Visual Studio Code or Installing the Amazon Toolkit for JetBrains.

Topics

Set up the testing and debugging environment

First, clone the Github repository that Amazon provides by entering the following command. The repository contains code samples for both Node.js canaries and Python canaries.

```
git clone https://github.com/aws-samples/synthetics-canary-local-debugging-sample.git
```

Then do one of the following, depending on the language of your canaries.

For Node.js canaries

1. Go to the Node.js canary source directory by entering the following command.

   ```
   cd synthetics-canary-local-debugging-sample/nodejs-canary/src
   ```

2. Enter the following command to install canary dependencies.

   ```
   npm install
   ```

For Python canaries

1. Go to the Python canary source directory by entering the following command.

```
cd synthetics-canary-local-debugging-sample/python-canary/src
```

2. Enter the following command to install canary dependencies.

```
pip3 install -r requirements.txt -t .
```

Use Visual Studio Code IDE

The Visual Studio launch configuration file is located at `.vscode/launch.json`. It contains configurations to allow the template file to be discovered by Visual Studio code. It defines a Lambda payload with required parameters to invoke the canary successfully. Here's the launch configuration for a Node.js canary:

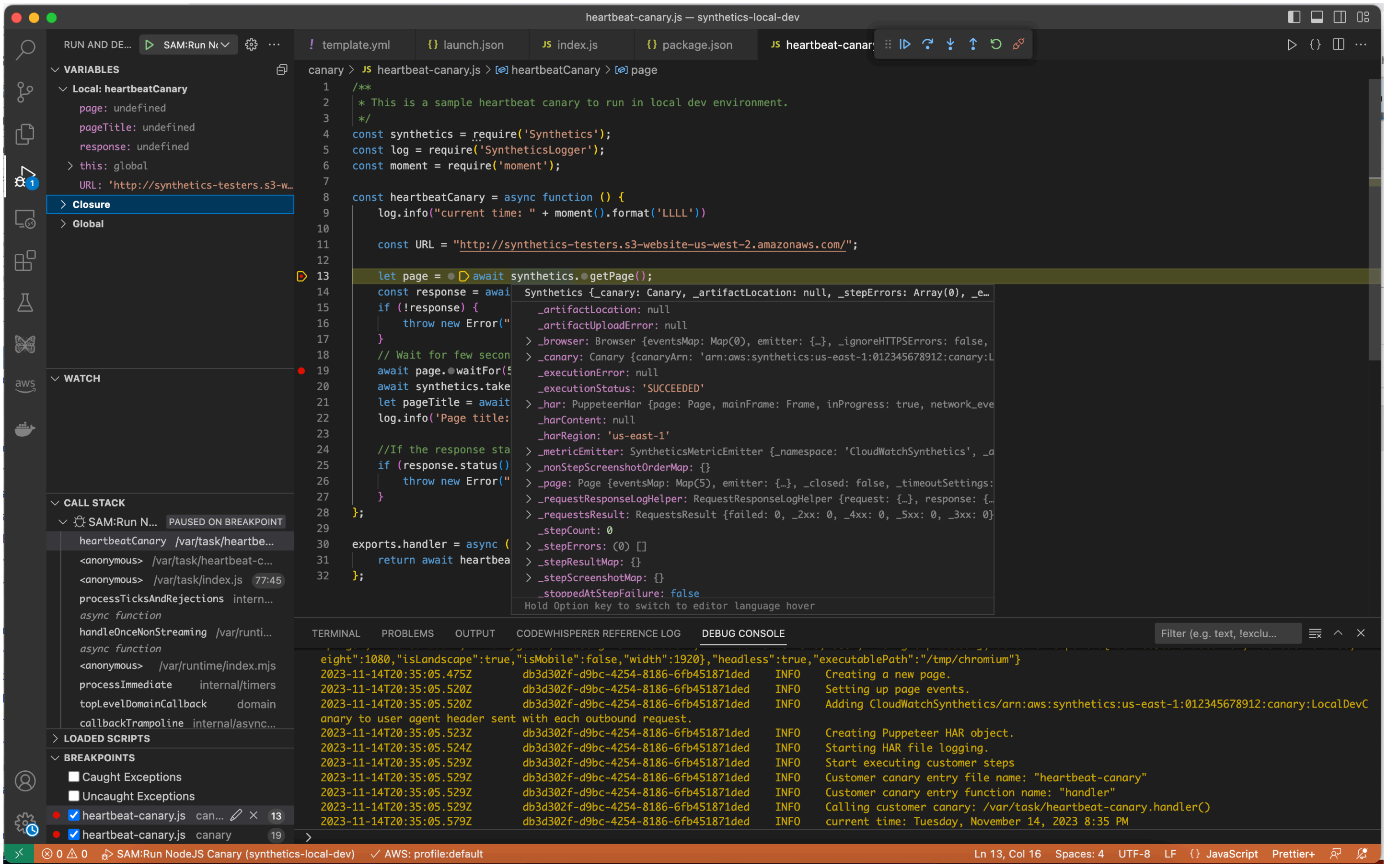
```
{
            ...
            ...
            "lambda": {
                "payload": {
                    "json": {
                        // Canary name. Provide any name you like.
                        "canaryName": "LocalSyntheticsCanary",
                        // Canary artifact location
                        "artifactS3Location": {
                            "s3Bucket": "cw-syn-results-123456789012-us-west-2",
                            "s3Key": "local-run-artifacts",
                        },
                        // Your canary handler name
                        "customerCanaryHandlerName": "heartbeat-canary.handler"
                    }
                },
                // Environment variables to pass to the canary code
                "environmentVariables": {}
            }
        }
    ]
}
```

You can also optionally provide the following fields in the payload JSON:

- `s3EncryptionMode` Valid values: `SSE_S3` | `SSE_KMS`
- `s3KmsKeyArn` Valid value: *`KMS Key ARN`*
- `activeTracing` Valid values: `true` | `false`
- `canaryRunId` Valid value: *`UUID`* This parameter is required if active tracing is enabled.

To debug the canary in Visual Studio, add breakpoints in the canary code where you want to pause execution. To add a breakpoint, choose the editor margin and go to **Run and Debug** mode in the editor. Run the canary by clicking on the play button. When the canary runs, the logs will be tailed in the debug console, providing you with real-time insights into the canary's behavior. If you added breakpoints, the canary execution will pause at each breakpoint, allowing you to step through code and inspect variable values, instance methods, object attributes, and the function call stack.

There is no cost incurred for running and debugging canaries locally, except for the artifacts stored in the Amazon S3 bucket and the CloudWatch metrics generated by each local run.



Use JetBrains IDE

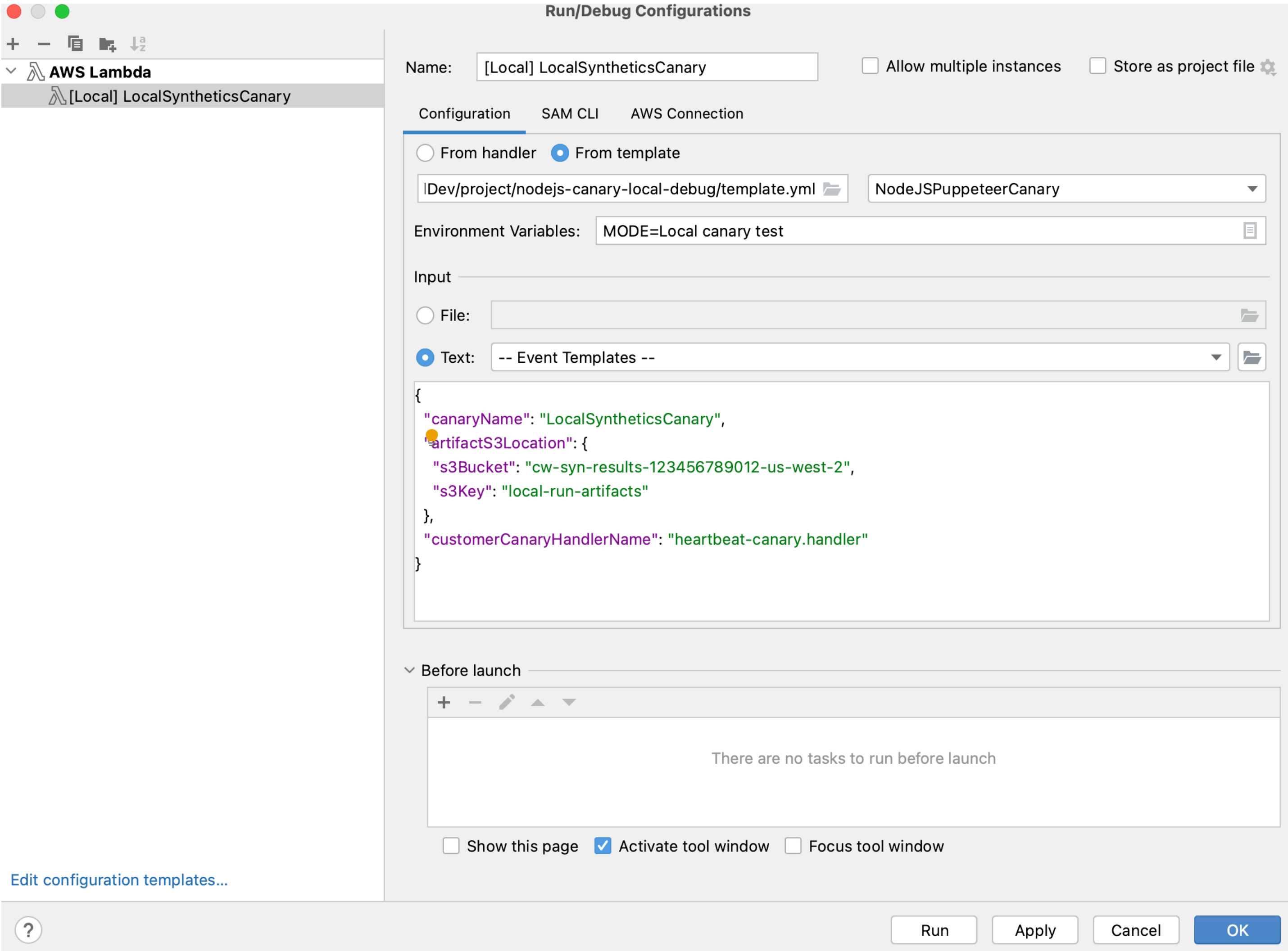
After you have the Amazon Toolkit for JetBrains extension installed, be sure that the Node.js plugin and JavaScript debugger are enabled to run, if you are debugging a Node.js canary. Then follow these steps.

Debug a canary using JetBrains IDE

1. In the left navigation pane of JetBrains IDE, choose **Lambda**, then choose the local configuration template.
2. Enter a name for the run configuration, such as **`LocalSyntheticsCanary`**
3. Choose **From template**, choose the file browser in the template field, then select the **template.yml** file from the project, either from the **nodejs** directory or the **python** directory.
4. In the **Input** section, enter the payload for the canary as shown in the following screen.

```
{
 "canaryName": "LocalSyntheticsCanary",
 "artifactS3Location": {
     "s3Bucket": "cw-syn-results-123456789012-us-west-2",
     "s3Key": "local-run-artifacts"
 },
 "customerCanaryHandlerName": "heartbeat-canary.handler"
}
```

   You can also set other environment variables in the payload JSON, as listed in Use Visual Studio Code IDE.



Run a canary locally with the SAM CLI

Use the one of the following procedures to run your canary locally using the Serverless Application Model (SAM) CLI. Be sure to specify your own Amazon S3 bucket name for `s3Bucket` in `event.json`

To use the SAM CLI to run a Node.js canary

1. Go to the source directory by entering the following command.

```
cd synthetics-canary-local-debugging-sample/nodejs-canary
```

2. Enter the following commands.

```
sam build
```

```
sam local invoke -e ../event.json
```

To use the SAM CLI to run a Python canary

1. Go to the source directory by entering the following command.

```
cd synthetics-canary-local-debugging-sample/python-canary
```

2. Enter the following commands.

```
sam build
sam local invoke -e ../event.json
```

Integrate your local testing environment into an existing canary package

You can integrate local canary debugging into your existing canary package by copying three files:

- Copy the `template.yml` file into your canary package root. Be sure to modify the path for `CodeUri` to point to the directory where your canary code exists.
- If you're working with a Node.js canary, copy the `cw-synthetics.js` file to your canary source directory. If you're working with a Python canary, copy the `cw-synthetics.py` to your canary source directory.
- Copy the launch configuration file `.vscode/launch.json` into the package root. Make sure to put it inside the `.vscode` directory; create it if it doesn't exist already.

Change the CloudWatch Synthetics runtime

As part of your debugging, you might want to try running a canary with a different CloudWatch Synthetics runtime, instead of the latest runtime. To do so, find the runtime that you want to use from one of the following tables. Be sure to select the runtime for the correct Region. Then paste the ARN for that runtime into the appropriate place in your `template.yml` file, and then run the canary.

Node.js and Puppeteer runtimes

ARNs for syn-nodejs-puppeteer-10.0

The following table lists the ARNs to use for version `syn-nodejs-puppeteer-10.0` of the CloudWatch Synthetics runtime in each Amazon Region where it is available.

Region	ARN
US East (N. Virginia)	`arn:aws:lambda:us-east-1:378653112637:layer:Synthetics:58`
US East (Ohio)	`arn:aws:lambda:us-east-2:772927465453:layer:Synthetics:61`
US West (N. California)	`arn:aws:lambda:us-west-1:332033056316:layer:Synthetics:59`
US West (Oregon)	`arn:aws:lambda:us-west-2:760325925879:layer:Synthetics:61`
Africa (Cape Town)	`arn:aws:lambda:af-south-1:461844272066:layer:Synthetics:59`
Asia Pacific (Hong Kong)	`arn:aws:lambda:ap-east-1:129828061636:layer:Synthetics:59`
Asia Pacific (Hyderabad)	`arn:aws:lambda:ap-south-2:280298676434:layer:Synthetics:34`
Asia Pacific (Jakarta)	`arn:aws:lambda:ap-southeast-3:246953257743:layer:Synthetics:41`
Asia Pacific (Malaysia)	`arn:aws:lambda:ap-southeast-5:035872523913:layer:Synthetics:15`
Asia Pacific (Melbourne)	`arn:aws:lambda:ap-southeast-4:200724813040:layer:Synthetics:32`

Region	ARN
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:724929286329:layer:Synthetics:59
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:608016332111:layer:Synthetics:45
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:989515803484:layer:Synthetics:62
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:068035103298:layer:Synthetics:63
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:584677157514:layer:Synthetics:58
Asia Pacific (Thailand)	arn:aws:lambda:ap-southeast-7:851725245975:layer:Synthetics:6
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:172291836251:layer:Synthetics:59
Canada (Central)	arn:aws:lambda:ca-central-1:236629016841:layer:Synthetics:59
Canada West (Calgary)	arn:aws:lambda:ca-west-1:944448206667:layer:Synthetics:90
China (Beijing)	arn:aws-cn:lambda:cn-north-1:422629156088:layer:Synthetics:58
China (Ningxia);	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:Synthetics:59
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:122305336817:layer:Synthetics:59

Region	ARN
Europe (Ireland)	arn:aws:lambda:eu-west-1:563204233543:layer:Synthetics:60
Europe (London)	arn:aws:lambda:eu-west-2:565831452869:layer:Synthetics:58
Europe (Milan)	arn:aws:lambda:eu-south-1:525618516618:layer:Synthetics:60
Europe (Paris)	arn:aws:lambda:eu-west-3:469466506258:layer:Synthetics:59
Europe (Spain)	arn:aws:lambda:eu-south-2:029793053121:layer:Synthetics:34
Europe (Stockholm)	arn:aws:lambda:eu-north-1:162938142733:layer:Synthetics:59
Europe (Zurich)	arn:aws:lambda:eu-central-2:224218992030:layer:Synthetics:33
Israel (Tel Aviv)	arn:aws:lambda:il-central-1:313249807427:layer:Synthetics:31
Mexico (Central)	arn:aws:lambda:mx-central-1:654654265476:layer:Synthetics:7
Middle East (Bahrain)	arn:aws:lambda:me-south-1:823195537320:layer:Synthetics:58
Middle East (UAE)	arn:aws:lambda:me-central-1:239544149032:layer:Synthetics:34
South America (São Paulo)	arn:aws:lambda:sa-east-1:783765544751:layer:Synthetics:60

Region	ARN
Amazon GovCloud (US-East)	`arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:Synthetics:54`
Amazon GovCloud (US-West)	`arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:Synthetics:55`

ARNs for syn-nodejs-puppeteer-9.1

The following table lists the ARNs to use for version `syn-nodejs-puppeteer-9.1` of the CloudWatch Synthetics runtime in each Amazon Region where it is available.

Region	ARN
US East (N. Virginia)	`arn:aws:lambda:us-east-1:378653112637:layer:Synthetics:53`
US East (Ohio)	`arn:aws:lambda:us-east-2:772927465453:layer:Synthetics:56`
US West (N. California)	`arn:aws:lambda:us-west-1:332033056316:layer:Synthetics:54`
US West (Oregon)	`arn:aws:lambda:us-west-2:760325925879:layer:Synthetics:56`
Africa (Cape Town)	`arn:aws:lambda:af-south-1:461844272066:layer:Synthetics:54`
Asia Pacific (Hong Kong)	`arn:aws:lambda:ap-east-1:129828061636:layer:Synthetics:54`
Asia Pacific (Hyderabad)	`arn:aws:lambda:ap-south-2:280298676434:layer:Synthetics:29`
Asia Pacific (Jakarta)	`arn:aws:lambda:ap-southeast-3:246953257743:layer:Synthetics:36`

Region	ARN
Asia Pacific (Malaysia)	`arn:aws:lambda:ap-southeast-5:035872523913:layer:Synthetics:10`
Asia Pacific (Melbourne)	`arn:aws:lambda:ap-southeast-4:200724813040:layer:Synthetics:27`
Asia Pacific (Mumbai)	`arn:aws:lambda:ap-south-1:724929286329:layer:Synthetics:54`
Asia Pacific (Osaka)	`arn:aws:lambda:ap-northeast-3:608016332111:layer:Synthetics:40`
Asia Pacific (Seoul)	`arn:aws:lambda:ap-northeast-2:989515803484:layer:Synthetics:57`
Asia Pacific (Singapore)	`arn:aws:lambda:ap-southeast-1:068035103298:layer:Synthetics:58`
Asia Pacific (Sydney)	`arn:aws:lambda:ap-southeast-2:584677157514:layer:Synthetics:53`
Asia Pacific (Thailand)	`arn:aws:lambda:ap-southeast-7:851725245975:layer:Synthetics:1`
Asia Pacific (Tokyo)	`arn:aws:lambda:ap-northeast-1:172291836251:layer:Synthetics:54`
Canada (Central)	`arn:aws:lambda:ca-central-1:236629016841:layer:Synthetics:54`
Canada West (Calgary)	`arn:aws:lambda:ca-west-1:944448206667:layer:Synthetics:85`
China (Beijing)	`arn:aws-cn:lambda:cn-north-1:422629156088:layer:Synthetics:54`

Region	ARN
China (Ningxia);	`arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:Synthetics:55`
Europe (Frankfurt)	`arn:aws:lambda:eu-central-1:122305336817:layer:Synthetics:54`
Europe (Ireland)	`arn:aws:lambda:eu-west-1:563204233543:layer:Synthetics:55`
Europe (London)	`arn:aws:lambda:eu-west-2:565831452869:layer:Synthetics:53`
Europe (Milan)	`arn:aws:lambda:eu-south-1:525618516618:layer:Synthetics:55`
Europe (Paris)	`arn:aws:lambda:eu-west-3:469466506258:layer:Synthetics:54`
Europe (Spain)	`arn:aws:lambda:eu-south-2:029793053121:layer:Synthetics:29`
Europe (Stockholm)	`arn:aws:lambda:eu-north-1:162938142733:layer:Synthetics:54`
Europe (Zurich)	`arn:aws:lambda:eu-central-2:224218992030:layer:Synthetics:28`
Israel (Tel Aviv)	`arn:aws:lambda:il-central-1:313249807427:layer:Synthetics:26`
Mexico (Central)	`arn:aws:lambda:mx-central-1:654654265476:layer:Synthetics:3`
Middle East (Bahrain)	`arn:aws:lambda:me-south-1:823195537320:layer:Synthetics:53`

Region	ARN
Middle East (UAE)	`arn:aws:lambda:me-central-1:239544149032:layer:Synthetics:29`
South America (São Paulo)	`arn:aws:lambda:sa-east-1:783765544751:layer:Synthetics:55`
Amazon GovCloud (US-East)	`arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:Synthetics:50`
Amazon GovCloud (US-West)	`arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:Synthetics:51`

ARNs for syn-nodejs-puppeteer-9.0

The following table lists the ARNs to use for version `syn-nodejs-puppeteer-9.0` of the CloudWatch Synthetics runtime in each Amazon Region where it is available.

Region	ARN
US East (N. Virginia)	`arn:aws:lambda:us-east-1:378653112637:layer:Synthetics:51`
US East (Ohio)	`arn:aws:lambda:us-east-2:772927465453:layer:Synthetics:54`
US West (N. California)	`arn:aws:lambda:us-west-1:332033056316:layer:Synthetics:52`
US West (Oregon)	`arn:aws:lambda:us-west-2:760325925879:layer:Synthetics:54`
Africa (Cape Town)	`arn:aws:lambda:af-south-1:461844272066:layer:Synthetics:52`
Asia Pacific (Hong Kong)	`arn:aws:lambda:ap-east-1:129828061636:layer:Synthetics:52`

Region	ARN
Asia Pacific (Hyderabad)	arn:aws:lambda:ap-south-2:280298676434:layer:Synthetics:27
Asia Pacific (Jakarta)	arn:aws:lambda:ap-southeast-3:246953257743:layer:Synthetics:34
Asia Pacific (Malaysia)	arn:aws:lambda:ap-southeast-5:035872523913:layer:Synthetics:8
Asia Pacific (Melbourne)	arn:aws:lambda:ap-southeast-4:200724813040:layer:Synthetics:25
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:724929286329:layer:Synthetics:52
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:608016332111:layer:Synthetics:38
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:989515803484:layer:Synthetics:55
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:068035103298:layer:Synthetics:56
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:584677157514:layer:Synthetics:51
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:172291836251:layer:Synthetics:52
Canada (Central)	arn:aws:lambda:ca-central-1:236629016841:layer:Synthetics:52
Canada West (Calgary)	arn:aws:lambda:ca-west-1:944448206667:layer:Synthetics:83

Region	ARN
China (Beijing)	arn:aws-cn:lambda:cn-north-1:422629156088:layer:Synthetics:52
China (Ningxia);	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:Synthetics:53
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:122305336817:layer:Synthetics:52
Europe (Ireland)	arn:aws:lambda:eu-west-1:563204233543:layer:Synthetics:53
Europe (London)	arn:aws:lambda:eu-west-2:565831452869:layer:Synthetics:51
Europe (Milan)	arn:aws:lambda:eu-south-1:525618516618:layer:Synthetics:53
Europe (Paris)	arn:aws:lambda:eu-west-3:469466506258:layer:Synthetics:52
Europe (Spain)	arn:aws:lambda:eu-south-2:029793053121:layer:Synthetics:27
Europe (Stockholm)	arn:aws:lambda:eu-north-1:162938142733:layer:Synthetics:52
Europe (Zurich)	arn:aws:lambda:eu-central-2:224218992030:layer:Synthetics:26
Israel (Tel Aviv)	arn:aws:lambda:il-central-1:313249807427:layer:Synthetics:24
Middle East (Bahrain)	arn:aws:lambda:me-south-1:823195537320:layer:Synthetics:51

Region	ARN
Middle East (UAE)	arn:aws:lambda:me-central-1:239544149032:layer:Synthetics:27
South America (São Paulo)	arn:aws:lambda:sa-east-1:783765544751:layer:Synthetics:53
Amazon GovCloud (US-East)	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:Synthetics:48
Amazon GovCloud (US-West)	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:Synthetics:49

ARNs for syn-nodejs-puppeteer-8.0

The following table lists the ARNs to use for version `syn-nodejs-puppeteer-8.0` of the CloudWatch Synthetics runtime in each Amazon Region where it is available.

Region	ARN
US East (N. Virginia)	arn:aws:lambda:us-east-1:378653112637:layer:Synthetics:48
US East (Ohio)	arn:aws:lambda:us-east-2:772927465453:layer:Synthetics:50
US West (N. California)	arn:aws:lambda:us-west-1:332033056316:layer:Synthetics:48
US West (Oregon)	arn:aws:lambda:us-west-2:760325925879:layer:Synthetics:51
Africa (Cape Town)	arn:aws:lambda:af-south-1:461844272066:layer:Synthetics:48
Asia Pacific (Hong Kong)	arn:aws:lambda:ap-east-1:129828061636:layer:Synthetics:49

Region	ARN
Asia Pacific (Hyderabad)	arn:aws:lambda:ap-south-2:280298676434:layer:Synthetics:24
Asia Pacific (Jakarta)	arn:aws:lambda:ap-southeast-3:246953257743:layer:Synthetics:30
Asia Pacific (Melbourne)	arn:aws:lambda:ap-southeast-4:200724813040:layer:Synthetics:22
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:724929286329:layer:Synthetics:48
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:608016332111:layer:Synthetics:34
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:989515803484:layer:Synthetics:51
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:068035103298:layer:Synthetics:53
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:584677157514:layer:Synthetics:48
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:172291836251:layer:Synthetics:48
Canada (Central)	arn:aws:lambda:ca-central-1:236629016841:layer:Synthetics:48
Canada West (Calgary)	arn:aws:lambda:ca-west-1:944448206667:layer:Synthetics:80
China (Beijing)	arn:aws-cn:lambda:cn-north-1:422629156088:layer:Synthetics:49

Region	ARN
China (Ningxia);	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:Synthetics:50
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:122305336817:layer:Synthetics:48
Europe (Ireland)	arn:aws:lambda:eu-west-1:563204233543:layer:Synthetics:50
Europe (London)	arn:aws:lambda:eu-west-2:565831452869:layer:Synthetics:48
Europe (Milan)	arn:aws:lambda:eu-south-1:525618516618:layer:Synthetics:49
Europe (Paris)	arn:aws:lambda:eu-west-3:469466506258:layer:Synthetics:48
Europe (Spain)	arn:aws:lambda:eu-south-2:029793053121:layer:Synthetics:24
Europe (Stockholm)	arn:aws:lambda:eu-north-1:162938142733:layer:Synthetics:48
Europe (Zurich)	arn:aws:lambda:eu-central-2:224218992030:layer:Synthetics:23
Israel (Tel Aviv)	arn:aws:lambda:il-central-1:313249807427:layer:Synthetics:21
Middle East (Bahrain)	arn:aws:lambda:me-south-1:823195537320:layer:Synthetics:48
Middle East (UAE)	arn:aws:lambda:me-central-1:239544149032:layer:Synthetics:23

Region	ARN
South America (São Paulo)	arn:aws:lambda:sa-east-1:783765544751:layer:Synthetics:49
Amazon GovCloud (US-East)	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:Synthetics:45
Amazon GovCloud (US-West)	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:Synthetics:46

ARNs for syn-nodejs-puppeteer-7.0

The following table lists the ARNs to use for version `syn-nodejs-puppeteer-7.0` of the CloudWatch Synthetics runtime in each Amazon Region where it is available.

Region	ARN
US East (N. Virginia)	arn:aws:lambda:us-east-1:378653112637:layer:Synthetics:44
US East (Ohio)	arn:aws:lambda:us-east-2:772927465453:layer:Synthetics:46
US West (N. California)	arn:aws:lambda:us-west-1:332033056316:layer:Synthetics:44
US West (Oregon)	arn:aws:lambda:us-west-2:760325925879:layer:Synthetics:47
Africa (Cape Town)	arn:aws:lambda:af-south-1:461844272066:layer:Synthetics:44
Asia Pacific (Hong Kong)	arn:aws:lambda:ap-east-1:129828061636:layer:Synthetics:45
Asia Pacific (Hyderabad)	arn:aws:lambda:ap-south-2:280298676434:layer:Synthetics:20

Region	ARN
Asia Pacific (Jakarta)	arn:aws:lambda:ap-southeast-3:246953257743:layer:Synthetics:26
Asia Pacific (Malaysia)	arn:aws:lambda:ap-southeast-5:035872523913:layer:Synthetics:7
Asia Pacific (Melbourne)	arn:aws:lambda:ap-southeast-4:200724813040:layer:Synthetics:18
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:724929286329:layer:Synthetics:44
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:608016332111:layer:Synthetics:30
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:989515803484:layer:Synthetics:46
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:068035103298:layer:Synthetics:49
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:584677157514:layer:Synthetics:44
Asia Pacific (Thailand)	arn:aws:lambda:ap-southeast-7:851725245975:layer:Synthetics:3
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:172291836251:layer:Synthetics:44
Canada (Central)	arn:aws:lambda:ca-central-1:236629016841:layer:Synthetics:44
Canada West (Calgary)	arn:aws:lambda:ca-west-1:944448206667:layer:Synthetics:76

Region	ARN
China (Beijing)	arn:aws-cn:lambda:cn-north-1:422629156088:layer:Synthetics:45
China (Ningxia);	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:Synthetics:46
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:122305336817:layer:Synthetics:44
Europe (Ireland)	arn:aws:lambda:eu-west-1:563204233543:layer:Synthetics:46
Europe (London)	arn:aws:lambda:eu-west-2:565831452869:layer:Synthetics:44
Europe (Milan)	arn:aws:lambda:eu-south-1:525618516618:layer:Synthetics:45
Europe (Paris)	arn:aws:lambda:eu-west-3:469466506258:layer:Synthetics:44
Europe (Spain)	arn:aws:lambda:eu-south-2:029793053121:layer:Synthetics:20
Europe (Stockholm)	arn:aws:lambda:eu-north-1:162938142733:layer:Synthetics:44
Europe (Zurich)	arn:aws:lambda:eu-central-2:224218992030:layer:Synthetics:19
Israel (Tel Aviv)	arn:aws:lambda:il-central-1:313249807427:layer:Synthetics:17
Mexico (Central)	arn:aws:lambda:mx-central-1:654654265476:layer:Synthetics:4

Region	ARN
Middle East (Bahrain)	arn:aws:lambda:me-south-1:823195537320:layer:Synthetics:44
Middle East (UAE)	arn:aws:lambda:me-central-1:239544149032:layer:Synthetics:19
South America (São Paulo)	arn:aws:lambda:sa-east-1:783765544751:layer:Synthetics:45
Amazon GovCloud (US-East)	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:Synthetics:41
Amazon GovCloud (US-West)	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:Synthetics:42

ARNs for syn-nodejs-puppeteer-6.2

The following table lists the ARNs to use for version `syn-nodejs-puppeteer-6.2` of the CloudWatch Synthetics runtime in each Amazon Region where it is available.

Region	ARN
US East (N. Virginia)	arn:aws:lambda:us-east-1:378653112637:layer:Synthetics:41
US East (Ohio)	arn:aws:lambda:us-east-2:772927465453:layer:Synthetics:43
US West (N. California)	arn:aws:lambda:us-west-1:332033056316:layer:Synthetics:41
US West (Oregon)	arn:aws:lambda:us-west-2:760325925879:layer:Synthetics:44
Africa (Cape Town)	arn:aws:lambda:af-south-1:461844272066:layer:Synthetics:41

Region	ARN
Asia Pacific (Hong Kong)	arn:aws:lambda:ap-east-1:129828061636:layer:Synthetics:42
Asia Pacific (Hyderabad)	arn:aws:lambda:ap-south-2:280298676434:layer:Synthetics:17
Asia Pacific (Jakarta)	arn:aws:lambda:ap-southeast-3:246953257743:layer:Synthetics:23
Asia Pacific (Melbourne)	arn:aws:lambda:ap-southeast-4:200724813040:layer:Synthetics:15
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:724929286329:layer:Synthetics:41
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:608016332111:layer:Synthetics:27
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:989515803484:layer:Synthetics:42
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:068035103298:layer:Synthetics:46
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:584677157514:layer:Synthetics:41
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:172291836251:layer:Synthetics:41
Canada (Central)	arn:aws:lambda:ca-central-1:236629016841:layer:Synthetics:41
Canada West (Calgary)	arn:aws:lambda:ca-west-1:944448206667:layer:Synthetics:73

Region	ARN
China (Beijing)	`arn:aws-cn:lambda:cn-north-1:422629156088:layer:Synthetics:42`
China (Ningxia);	`arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:Synthetics:43`
Europe (Frankfurt)	`arn:aws:lambda:eu-central-1:122305336817:layer:Synthetics:41`
Europe (Ireland)	`arn:aws:lambda:eu-west-1:563204233543:layer:Synthetics:43`
Europe (London)	`arn:aws:lambda:eu-west-2:565831452869:layer:Synthetics:41`
Europe (Milan)	`arn:aws:lambda:eu-south-1:525618516618:layer:Synthetics:42`
Europe (Paris)	`arn:aws:lambda:eu-west-3:469466506258:layer:Synthetics:41`
Europe (Spain)	`arn:aws:lambda:eu-south-2:029793053121:layer:Synthetics:17`
Europe (Stockholm)	`arn:aws:lambda:eu-north-1:162938142733:layer:Synthetics:41`
Europe (Zurich)	`arn:aws:lambda:eu-central-2:224218992030:layer:Synthetics:16`
Israel (Tel Aviv)	`arn:aws:lambda:il-central-1:313249807427:layer:Synthetics:14`
Middle East (Bahrain)	`arn:aws:lambda:me-south-1:823195537320:layer:Synthetics:41`

Region	ARN
Middle East (UAE)	`arn:aws:lambda:me-central-1:239544149032:layer:Synthetics:16`
South America (São Paulo)	`arn:aws:lambda:sa-east-1:783765544751:layer:Synthetics:42`
Amazon GovCloud (US-East)	`arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:Synthetics:39`
Amazon GovCloud (US-West)	`arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:Synthetics:39`

ARNs for syn-nodejs-puppeteer-5.2

The following table lists the ARNs to use for version `syn-nodejs-puppeteer-5.2` of the CloudWatch Synthetics runtime in each Amazon Region where it is available.

Region	ARN
US East (N. Virginia)	`arn:aws:lambda:us-east-1:378653112637:layer:Synthetics:42`
US East (Ohio)	`arn:aws:lambda:us-east-2:772927465453:layer:Synthetics:44`
US West (N. California)	`arn:aws:lambda:us-west-1:332033056316:layer:Synthetics:42`
US West (Oregon)	`arn:aws:lambda:us-west-2:760325925879:layer:Synthetics:45`
Africa (Cape Town)	`arn:aws:lambda:af-south-1:461844272066:layer:Synthetics:42`
Asia Pacific (Hong Kong)	`arn:aws:lambda:ap-east-1:129828061636:layer:Synthetics:43`

Region	ARN
Asia Pacific (Hyderabad)	arn:aws:lambda:ap-south-2:280298676434:layer:Synthetics:18
Asia Pacific (Jakarta)	arn:aws:lambda:ap-southeast-3:246953257743:layer:Synthetics:24
Asia Pacific (Melbourne)	arn:aws:lambda:ap-southeast-4:200724813040:layer:Synthetics:16
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:724929286329:layer:Synthetics:42
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:608016332111:layer:Synthetics:28
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:989515803484:layer:Synthetics:44
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:068035103298:layer:Synthetics:47
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:584677157514:layer:Synthetics:42
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:172291836251:layer:Synthetics:42
Canada (Central)	arn:aws:lambda:ca-central-1:236629016841:layer:Synthetics:42
Canada West (Calgary)	arn:aws:lambda:ca-west-1:944448206667:layer:Synthetics:74
China (Beijing)	arn:aws-cn:lambda:cn-north-1:422629156088:layer:Synthetics:43

Region	ARN
China (Ningxia);	`arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:Synthetics:44`
Europe (Frankfurt)	`arn:aws:lambda:eu-central-1:122305336817:layer:Synthetics:42`
Europe (Ireland)	`arn:aws:lambda:eu-west-1:563204233543:layer:Synthetics:44`
Europe (London)	`arn:aws:lambda:eu-west-2:565831452869:layer:Synthetics:42`
Europe (Milan)	`arn:aws:lambda:eu-south-1:525618516618:layer:Synthetics:43`
Europe (Paris)	`arn:aws:lambda:eu-west-3:469466506258:layer:Synthetics:42`
Europe (Spain)	`arn:aws:lambda:eu-south-2:029793053121:layer:Synthetics:18`
Europe (Stockholm)	`arn:aws:lambda:eu-north-1:162938142733:layer:Synthetics:42`
Europe (Zurich)	`arn:aws:lambda:eu-central-2:224218992030:layer:Synthetics:17`
Israel (Tel Aviv)	`arn:aws:lambda:il-central-1:313249807427:layer:Synthetics:15`
Middle East (Bahrain)	`arn:aws:lambda:me-south-1:823195537320:layer:Synthetics:42`
Middle East (UAE)	`arn:aws:lambda:me-central-1:239544149032:layer:Synthetics:17`

Region	ARN
South America (São Paulo)	`arn:aws:lambda:sa-east-1:783765544751:layer:Synthetics:43`
Amazon GovCloud (US-East)	`arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:Synthetics:40`
Amazon GovCloud (US-West)	`arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:Synthetics:40`

Node.js and Playwright runtimes

ARNs for syn-nodejs-playwright-2.0

The following table lists the ARNs to use for version `syn-nodejs-playwright-2.0` of the CloudWatch Synthetics runtime in each Amazon Region where it is available.

Region	ARN
US East (N. Virginia)	`arn:aws:lambda:us-east-1:378653112637:layer:AWS-CW-SyntheticsNodeJsPlaywright:4`
US East (Ohio)	`arn:aws:lambda:us-east-2:772927465453:layer:AWS-CW-SyntheticsNodeJsPlaywright:4`
US West (N. California)	`arn:aws:lambda:us-west-1:332033056316:layer:AWS-CW-SyntheticsNodeJsPlaywright:4`
US West (Oregon)	`arn:aws:lambda:us-west-2:760325925879:layer:AWS-CW-SyntheticsNodeJsPlaywright:4`
Africa (Cape Town)	`arn:aws:lambda:af-south-1:461844272066:layer:AWS-CW-SyntheticsNodeJsPlaywright:4`
Asia Pacific (Hong Kong)	`arn:aws:lambda:ap-east-1:129828061636:layer:AWS-CW-SyntheticsNodeJsPlaywright:4`

Region	ARN
Asia Pacific (Hyderabad)	arn:aws:lambda:ap-south-2:280298676434:layer:AWS-CW-SyntheticsNodeJsPlaywright:4
Asia Pacific (Jakarta)	arn:aws:lambda:ap-southeast-3:246953257743:layer:AWS-CW-SyntheticsNodeJsPlaywright:4
Asia Pacific (Malaysia)	arn:aws:lambda:ap-southeast-5:035872523913:layer:AWS-CW-SyntheticsNodeJsPlaywright:4
Asia Pacific (Melbourne)	arn:aws:lambda:ap-southeast-4:200724813040:layer:AWS-CW-SyntheticsNodeJsPlaywright:4
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:724929286329:layer:AWS-CW-SyntheticsNodeJsPlaywright:4
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:608016332111:layer:AWS-CW-SyntheticsNodeJsPlaywright:4
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:989515803484:layer:AWS-CW-SyntheticsNodeJsPlaywright:4
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:068035103298:layer:AWS-CW-SyntheticsNodeJsPlaywright:4
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:584677157514:layer:AWS-CW-SyntheticsNodeJsPlaywright:4
Asia Pacific (Thailand)	arn:aws:lambda:ap-southeast-7:851725245975:layer:AWS-CW-SyntheticsNodeJsPlaywright:3
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:172291836251:layer:AWS-CW-SyntheticsNodeJsPlaywright:4
Canada (Central)	arn:aws:lambda:ca-central-1:236629016841:layer:AWS-CW-SyntheticsNodeJsPlaywright:4

Region	ARN
Canada West (Calgary)	arn:aws:lambda:ca-west-1:944448206667:layer:AWS-CW-SyntheticsNodeJsPlaywright:4
China (Beijing)	arn:aws-cn:lambda:cn-north-1:422629156088:layer:AWS-CW-SyntheticsNodeJsPlaywright:3
China (Ningxia);	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:AWS-CW-SyntheticsNodeJsPlaywright:3
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:122305336817:layer:AWS-CW-SyntheticsNodeJsPlaywright:4
Europe (Ireland)	arn:aws:lambda:eu-west-1:563204233543:layer:AWS-CW-SyntheticsNodeJsPlaywright:4
Europe (London)	arn:aws:lambda:eu-west-2:565831452869:layer:AWS-CW-SyntheticsNodeJsPlaywright:4
Europe (Milan)	arn:aws:lambda:eu-south-1:525618516618:layer:AWS-CW-SyntheticsNodeJsPlaywright:4
Europe (Paris)	arn:aws:lambda:eu-west-3:469466506258:layer:AWS-CW-SyntheticsNodeJsPlaywright:4
Europe (Spain)	arn:aws:lambda:eu-south-2:029793053121:layer:AWS-CW-SyntheticsNodeJsPlaywright:4
Europe (Stockholm)	arn:aws:lambda:eu-north-1:162938142733:layer:AWS-CW-SyntheticsNodeJsPlaywright:4
Europe (Zurich)	arn:aws:lambda:eu-central-2:224218992030:layer:AWS-CW-SyntheticsNodeJsPlaywright:4
Israel (Tel Aviv)	arn:aws:lambda:il-central-1:313249807427:layer:AWS-CW-SyntheticsNodeJsPlaywright:4

Region	ARN
Mexico (Central)	arn:aws:lambda:mx-central-1:654654265476:layer:AWS-CW-SyntheticsNodeJsPlaywright:5
Middle East (Bahrain)	arn:aws:lambda:me-south-1:823195537320:layer:AWS-CW-SyntheticsNodeJsPlaywright:4
Middle East (UAE)	arn:aws:lambda:me-central-1:239544149032:layer:AWS-CW-SyntheticsNodeJsPlaywright:4
South America (São Paulo)	arn:aws:lambda:sa-east-1:783765544751:layer:AWS-CW-SyntheticsNodeJsPlaywright:4
Amazon GovCloud (US-East)	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:AWS-CW-SyntheticsNodeJsPlaywright:3
Amazon GovCloud (US-West)	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:AWS-CW-SyntheticsNodeJsPlaywright:3

ARNs for syn-nodejs-playwright-1.0

The following table lists the ARNs to use for version `syn-nodejs-playwright-1.0` of the CloudWatch Synthetics runtime in each Amazon Region where it is available.

Region	ARN
US East (N. Virginia)	arn:aws:lambda:us-east-1:378653112637:layer:AWS-CW-SyntheticsNodeJsPlaywright:1
US East (Ohio)	arn:aws:lambda:us-east-2:772927465453:layer:AWS-CW-SyntheticsNodeJsPlaywright:1
US West (N. California)	arn:aws:lambda:us-west-1:332033056316:layer:AWS-CW-SyntheticsNodeJsPlaywright:1
US West (Oregon)	arn:aws:lambda:us-west-2:760325925879:layer:AWS-CW-SyntheticsNodeJsPlaywright:1

Region	ARN
Africa (Cape Town)	arn:aws:lambda:af-south-1:461844272066:layer:AWS-CW-SyntheticsNodeJsPlaywright:1
Asia Pacific (Hong Kong)	arn:aws:lambda:ap-east-1:129828061636:layer:AWS-CW-SyntheticsNodeJsPlaywright:1
Asia Pacific (Hyderabad)	arn:aws:lambda:ap-south-2:280298676434:layer:AWS-CW-SyntheticsNodeJsPlaywright:1
Asia Pacific (Jakarta)	arn:aws:lambda:ap-southeast-3:246953257743:layer:AWS-CW-SyntheticsNodeJsPlaywright:1
Asia Pacific (Malaysia)	arn:aws:lambda:ap-southeast-5:035872523913:layer:AWS-CW-SyntheticsNodeJsPlaywright:1
Asia Pacific (Melbourne)	arn:aws:lambda:ap-southeast-4:200724813040:layer:AWS-CW-SyntheticsNodeJsPlaywright:1
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:724929286329:layer:AWS-CW-SyntheticsNodeJsPlaywright:1
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:608016332111:layer:AWS-CW-SyntheticsNodeJsPlaywright:1
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:989515803484:layer:AWS-CW-SyntheticsNodeJsPlaywright:1
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:068035103298:layer:AWS-CW-SyntheticsNodeJsPlaywright:1
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:584677157514:layer:AWS-CW-SyntheticsNodeJsPlaywright:1
Asia Pacific (Thailand)	arn:aws:lambda:ap-southeast-7:851725245975:layer:AWS-CW-SyntheticsNodeJsPlaywright:1

Region	ARN
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:172291836251:layer:AWS-CW-SyntheticsNodeJsPlaywright:1
Canada (Central)	arn:aws:lambda:ca-central-1:236629016841:layer:AWS-CW-SyntheticsNodeJsPlaywright:1
Canada West (Calgary)	arn:aws:lambda:ca-west-1:944448206667:layer:AWS-CW-SyntheticsNodeJsPlaywright:1
China (Beijing)	arn:aws-cn:lambda:cn-north-1:422629156088:layer:AWS-CW-SyntheticsNodeJsPlaywright:1
China (Ningxia);	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:AWS-CW-SyntheticsNodeJsPlaywright:1
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:122305336817:layer:AWS-CW-SyntheticsNodeJsPlaywright:1
Europe (Ireland)	arn:aws:lambda:eu-west-1:563204233543:layer:AWS-CW-SyntheticsNodeJsPlaywright:1
Europe (London)	arn:aws:lambda:eu-west-2:565831452869:layer:AWS-CW-SyntheticsNodeJsPlaywright:1
Europe (Milan)	arn:aws:lambda:eu-south-1:525618516618:layer:AWS-CW-SyntheticsNodeJsPlaywright:1
Europe (Paris)	arn:aws:lambda:eu-west-3:469466506258:layer:AWS-CW-SyntheticsNodeJsPlaywright:1
Europe (Spain)	arn:aws:lambda:eu-south-2:029793053121:layer:AWS-CW-SyntheticsNodeJsPlaywright:1
Europe (Stockholm)	arn:aws:lambda:eu-north-1:162938142733:layer:AWS-CW-SyntheticsNodeJsPlaywright:1

Region	ARN
Europe (Zurich)	arn:aws:lambda:eu-central-2:224218992030:layer:AWS-CW-SyntheticsNodeJsPlaywright:1
Israel (Tel Aviv)	arn:aws:lambda:il-central-1:313249807427:layer:AWS-CW-SyntheticsNodeJsPlaywright:1
Mexico (Central)	arn:aws:lambda:mx-central-1:654654265476:layer:AWS-CW-SyntheticsNodeJsPlaywright:3
Middle East (Bahrain)	arn:aws:lambda:me-south-1:823195537320:layer:AWS-CW-SyntheticsNodeJsPlaywright:1
Middle East (UAE)	arn:aws:lambda:me-central-1:239544149032:layer:AWS-CW-SyntheticsNodeJsPlaywright:1
South America (São Paulo)	arn:aws:lambda:sa-east-1:783765544751:layer:AWS-CW-SyntheticsNodeJsPlaywright:1
Amazon GovCloud (US-East)	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:AWS-CW-SyntheticsNodeJsPlaywright:1
Amazon GovCloud (US-West)	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:AWS-CW-SyntheticsNodeJsPlaywright:1

Python and Selenium runtimes

ARNs for syn-python-selenium-5.1

The following table lists the ARNs to use for version `syn-python-selenium-5.1` of the CloudWatch Synthetics runtime in each Amazon Region where it is available.

Region	ARN
US East (N. Virginia)	arn:aws:lambda:us-east-1:378653112637:layer:Synthetics_Selenium:45

Region	ARN
US East (Ohio)	arn:aws:lambda:us-east-2:772927465453:layer:Synthetics_Selenium:48
US West (N. California)	arn:aws:lambda:us-west-1:332033056316:layer:Synthetics_Selenium:46
US West (Oregon)	arn:aws:lambda:us-west-2:760325925879:layer:Synthetics_Selenium:47
Africa (Cape Town)	arn:aws:lambda:af-south-1:461844272066:layer:Synthetics_Selenium:46
Asia Pacific (Hong Kong)	arn:aws:lambda:ap-east-1:129828061636:layer:Synthetics_Selenium:45
Asia Pacific (Hyderabad)	arn:aws:lambda:ap-south-2:280298676434:layer:Synthetics_Selenium:33
Asia Pacific (Jakarta)	arn:aws:lambda:ap-southeast-3:246953257743:layer:Synthetics_Selenium:40
Asia Pacific (Malaysia)	arn:aws:lambda:ap-southeast-5:035872523913:layer:Synthetics_Selenium:15
Asia Pacific (Melbourne)	arn:aws:lambda:ap-southeast-4:200724813040:layer:Synthetics_Selenium:31
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:724929286329:layer:Synthetics_Selenium:46
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:608016332111:layer:Synthetics_Selenium:44
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:989515803484:layer:Synthetics_Selenium:49

Region	ARN
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:068035103298:layer:Synthetics_Selenium:50
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:584677157514:layer:Synthetics_Selenium:45
Asia Pacific (Thailand)	arn:aws:lambda:ap-southeast-7:851725245975:layer:Synthetics_Selenium:6
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:172291836251:layer:Synthetics_Selenium:46
Canada (Central)	arn:aws:lambda:ca-central-1:236629016841:layer:Synthetics_Selenium:44
Canada West (Calgary)	arn:aws:lambda:ca-west-1:944448206667:layer:Synthetics_Selenium:89
China (Beijing)	arn:aws-cn:lambda:cn-north-1:422629156088:layer:Synthetics_Selenium:44
China (Ningxia);	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:Synthetics_Selenium:44
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:122305336817:layer:Synthetics_Selenium:46
Europe (Ireland)	arn:aws:lambda:eu-west-1:563204233543:layer:Synthetics_Selenium:47
Europe (London)	arn:aws:lambda:eu-west-2:565831452869:layer:Synthetics_Selenium:45
Europe (Milan)	arn:aws:lambda:eu-south-1:525618516618:layer:Synthetics_Selenium:47

Region	ARN
Europe (Paris)	arn:aws:lambda:eu-west-3:469466506258:layer:Synthetics_Selenium:46
Europe (Spain)	arn:aws:lambda:eu-south-2:029793053121:layer:Synthetics_Selenium:33
Europe (Stockholm)	arn:aws:lambda:eu-north-1:162938142733:layer:Synthetics_Selenium:46
Europe (Zurich)	arn:aws:lambda:eu-central-2:224218992030:layer:Synthetics_Selenium:32
Israel (Tel Aviv)	arn:aws:lambda:il-central-1:313249807427:layer:Synthetics_Selenium:30
Mexico (Central)	arn:aws:lambda:mx-central-1:654654265476:layer:Synthetics_Selenium:7
Middle East (Bahrain)	arn:aws:lambda:me-south-1:823195537320:layer:Synthetics_Selenium:45
Middle East (UAE)	arn:aws:lambda:me-central-1:239544149032:layer:Synthetics_Selenium:33
South America (São Paulo)	arn:aws:lambda:sa-east-1:783765544751:layer:Synthetics_Selenium:47
Amazon GovCloud (US-East)	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:Synthetics_Selenium:42
Amazon GovCloud (US-West)	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:Synthetics_Selenium:43

ARNs for syn-python-selenium-5.0

The following table lists the ARNs to use for version `syn-python-selenium-5.0` of the CloudWatch Synthetics runtime in each Amazon Region where it is available.

Region	ARN
US East (N. Virginia)	`arn:aws:lambda:us-east-1:378653112637:layer:Synthetics_Selenium:43`
US East (Ohio)	`arn:aws:lambda:us-east-2:772927465453:layer:Synthetics_Selenium:46`
US West (N. California)	`arn:aws:lambda:us-west-1:332033056316:layer:Synthetics_Selenium:44`
US West (Oregon)	`arn:aws:lambda:us-west-2:760325925879:layer:Synthetics_Selenium:45`
Africa (Cape Town)	`arn:aws:lambda:af-south-1:461844272066:layer:Synthetics_Selenium:44`
Asia Pacific (Hong Kong)	`arn:aws:lambda:ap-east-1:129828061636:layer:Synthetics_Selenium:43`
Asia Pacific (Hyderabad)	`arn:aws:lambda:ap-south-2:280298676434:layer:Synthetics_Selenium:31`
Asia Pacific (Jakarta)	`arn:aws:lambda:ap-southeast-3:246953257743:layer:Synthetics_Selenium:38`
Asia Pacific (Malaysia)	`arn:aws:lambda:ap-southeast-5:035872523913:layer:Synthetics_Selenium:13`
Asia Pacific (Melbourne)	`arn:aws:lambda:ap-southeast-4:200724813040:layer:Synthetics_Selenium:29`
Asia Pacific (Mumbai)	`arn:aws:lambda:ap-south-1:724929286329:layer:Synthetics_Selenium:44`
Asia Pacific (Osaka)	`arn:aws:lambda:ap-northeast-3:608016332111:layer:Synthetics_Selenium:42`

Region	ARN
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:989515803484:layer:Synthetics_Selenium:47
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:068035103298:layer:Synthetics_Selenium:48
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:584677157514:layer:Synthetics_Selenium:43
Asia Pacific (Thailand)	arn:aws:lambda:ap-southeast-7:851725245975:layer:Synthetics_Selenium:4
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:172291836251:layer:Synthetics_Selenium:44
Canada (Central)	arn:aws:lambda:ca-central-1:236629016841:layer:Synthetics_Selenium:44
Canada West (Calgary)	arn:aws:lambda:ca-west-1:944448206667:layer:Synthetics_Selenium:87
China (Beijing)	arn:aws-cn:lambda:cn-north-1:422629156088:layer:Synthetics_Selenium:43
China (Ningxia);	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:Synthetics_Selenium:43
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:122305336817:layer:Synthetics_Selenium:44
Europe (Ireland)	arn:aws:lambda:eu-west-1:563204233543:layer:Synthetics_Selenium:45
Europe (London)	arn:aws:lambda:eu-west-2:565831452869:layer:Synthetics_Selenium:43

Region	ARN
Europe (Milan)	arn:aws:lambda:eu-south-1:525618516618:layer:Synthetics_Selenium:45
Europe (Paris)	arn:aws:lambda:eu-west-3:469466506258:layer:Synthetics_Selenium:44
Europe (Spain)	arn:aws:lambda:eu-south-2:029793053121:layer:Synthetics_Selenium:31
Europe (Stockholm)	arn:aws:lambda:eu-north-1:162938142733:layer:Synthetics_Selenium:44
Europe (Zurich)	arn:aws:lambda:eu-central-2:224218992030:layer:Synthetics_Selenium:30
Israel (Tel Aviv)	arn:aws:lambda:il-central-1:313249807427:layer:Synthetics_Selenium:28
Mexico (Central)	arn:aws:lambda:mx-central-1:654654265476:layer:Synthetics_Selenium:5
Middle East (Bahrain)	arn:aws:lambda:me-south-1:823195537320:layer:Synthetics_Selenium:43
Middle East (UAE)	arn:aws:lambda:me-central-1:239544149032:layer:Synthetics_Selenium:31
South America (São Paulo)	arn:aws:lambda:sa-east-1:783765544751:layer:Synthetics_Selenium:45
Amazon GovCloud (US-East)	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:Synthetics_Selenium:41
Amazon GovCloud (US-West)	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:Synthetics_Selenium:42

ARNs for syn-python-selenium-4.1

The following table lists the ARNs to use for version `syn-python-selenium-4.1` of the CloudWatch Synthetics runtime in each Amazon Region where it is available.

Region	ARN
US East (N. Virginia)	`arn:aws:lambda:us-east-1:378653112637:layer:Synthetics_Selenium:40`
US East (Ohio)	`arn:aws:lambda:us-east-2:772927465453:layer:Synthetics_Selenium:43`
US West (N. California)	`arn:aws:lambda:us-west-1:332033056316:layer:Synthetics_Selenium:41`
US West (Oregon)	`arn:aws:lambda:us-west-2:760325925879:layer:Synthetics_Selenium:42`
Africa (Cape Town)	`arn:aws:lambda:af-south-1:461844272066:layer:Synthetics_Selenium:41`
Asia Pacific (Hong Kong)	`arn:aws:lambda:ap-east-1:129828061636:layer:Synthetics_Selenium:40`
Asia Pacific (Hyderabad)	`arn:aws:lambda:ap-south-2:280298676434:layer:Synthetics_Selenium:28`
Asia Pacific (Jakarta)	`arn:aws:lambda:ap-southeast-3:246953257743:layer:Synthetics_Selenium:35`
Asia Pacific (Malaysia)	`arn:aws:lambda:ap-southeast-5:035872523913:layer:Synthetics_Selenium:10`
Asia Pacific (Melbourne)	`arn:aws:lambda:ap-southeast-4:200724813040:layer:Synthetics_Selenium:26`
Asia Pacific (Mumbai)	`arn:aws:lambda:ap-south-1:724929286329:layer:Synthetics_Selenium:41`

Region	ARN
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:608016332111:layer:Synthetics_Selenium:39
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:989515803484:layer:Synthetics_Selenium:44
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:068035103298:layer:Synthetics_Selenium:45
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:584677157514:layer:Synthetics_Selenium:40
Asia Pacific (Thailand)	arn:aws:lambda:ap-southeast-7:851725245975:layer:Synthetics_Selenium:1
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:172291836251:layer:Synthetics_Selenium:41
Canada (Central)	arn:aws:lambda:ca-central-1:236629016841:layer:Synthetics_Selenium:41
Canada West (Calgary)	arn:aws:lambda:ca-west-1:944448206667:layer:Synthetics_Selenium:84
China (Beijing)	arn:aws-cn:lambda:cn-north-1:422629156088:layer:Synthetics_Selenium:40
China (Ningxia);	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:Synthetics_Selenium:40
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:122305336817:layer:Synthetics_Selenium:41
Europe (Ireland)	arn:aws:lambda:eu-west-1:563204233543:layer:Synthetics_Selenium:42

Region	ARN
Europe (London)	arn:aws:lambda:eu-west-2:565831452869:layer:Synthetics_Selenium:40
Europe (Milan)	arn:aws:lambda:eu-south-1:525618516618:layer:Synthetics_Selenium:42
Europe (Paris)	arn:aws:lambda:eu-west-3:469466506258:layer:Synthetics_Selenium:41
Europe (Spain)	arn:aws:lambda:eu-south-2:029793053121:layer:Synthetics_Selenium:28
Europe (Stockholm)	arn:aws:lambda:eu-north-1:162938142733:layer:Synthetics_Selenium:41
Europe (Zurich)	arn:aws:lambda:eu-central-2:224218992030:layer:Synthetics_Selenium:27
Israel (Tel Aviv)	arn:aws:lambda:il-central-1:313249807427:layer:Synthetics_Selenium:25
Mexico (Central)	arn:aws:lambda:mx-central-1:654654265476:layer:Synthetics_Selenium:3
Middle East (Bahrain)	arn:aws:lambda:me-south-1:823195537320:layer:Synthetics_Selenium:40
Middle East (UAE)	arn:aws:lambda:me-central-1:239544149032:layer:Synthetics_Selenium:28
South America (São Paulo)	arn:aws:lambda:sa-east-1:783765544751:layer:Synthetics_Selenium:42
Amazon GovCloud (US-East)	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:Synthetics_Selenium:38

Region	ARN
Amazon GovCloud (US-West)	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:Synthetics_Selenium:39

ARNs for syn-python-selenium-4.0

The following table lists the ARNs to use for version `syn-python-selenium-4.0` of the CloudWatch Synthetics runtime in each Amazon Region where it is available.

Region	ARN
US East (N. Virginia)	arn:aws:lambda:us-east-1:378653112637:layer:Synthetics_Selenium:38
US East (Ohio)	arn:aws:lambda:us-east-2:772927465453:layer:Synthetics_Selenium:41
US West (N. California)	arn:aws:lambda:us-west-1:332033056316:layer:Synthetics_Selenium:39
US West (Oregon)	arn:aws:lambda:us-west-2:760325925879:layer:Synthetics_Selenium:40
Africa (Cape Town)	arn:aws:lambda:af-south-1:461844272066:layer:Synthetics_Selenium:39
Asia Pacific (Hong Kong)	arn:aws:lambda:ap-east-1:129828061636:layer:Synthetics_Selenium:38
Asia Pacific (Hyderabad)	arn:aws:lambda:ap-south-2:280298676434:layer:Synthetics_Selenium:26
Asia Pacific (Jakarta)	arn:aws:lambda:ap-southeast-3:246953257743:layer:Synthetics_Selenium:33
Asia Pacific (Malaysia)	arn:aws:lambda:ap-southeast-5:035872523913:layer:Synthetics_Selenium:8

Region	ARN
Asia Pacific (Melbourne)	arn:aws:lambda:ap-southeast-4:200724813040:layer:Synthetics_Selenium:24
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:724929286329:layer:Synthetics_Selenium:39
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:608016332111:layer:Synthetics_Selenium:37
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:989515803484:layer:Synthetics_Selenium:42
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:068035103298:layer:Synthetics_Selenium:43
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:584677157514:layer:Synthetics_Selenium:38
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:172291836251:layer:Synthetics_Selenium:39
Canada (Central)	arn:aws:lambda:ca-central-1:236629016841:layer:Synthetics_Selenium:39
Canada West (Calgary)	arn:aws:lambda:ca-west-1:944448206667:layer:Synthetics_Selenium:82
China (Beijing)	arn:aws-cn:lambda:cn-north-1:422629156088:layer:Synthetics_Selenium:38
China (Ningxia);	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:Synthetics_Selenium:38
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:122305336817:layer:Synthetics_Selenium:39

Region	ARN
Europe (Ireland)	arn:aws:lambda:eu-west-1:563204233543:layer:Synthetics_Selenium:40
Europe (London)	arn:aws:lambda:eu-west-2:565831452869:layer:Synthetics_Selenium:38
Europe (Milan)	arn:aws:lambda:eu-south-1:525618516618:layer:Synthetics_Selenium:40
Europe (Paris)	arn:aws:lambda:eu-west-3:469466506258:layer:Synthetics_Selenium:39
Europe (Spain)	arn:aws:lambda:eu-south-2:029793053121:layer:Synthetics_Selenium:26
Europe (Stockholm)	arn:aws:lambda:eu-north-1:162938142733:layer:Synthetics_Selenium:39
Europe (Zurich)	arn:aws:lambda:eu-central-2:224218992030:layer:Synthetics_Selenium:25
Israel (Tel Aviv)	arn:aws:lambda:il-central-1:313249807427:layer:Synthetics_Selenium:23
Middle East (Bahrain)	arn:aws:lambda:me-south-1:823195537320:layer:Synthetics_Selenium:38
Middle East (UAE)	arn:aws:lambda:me-central-1:239544149032:layer:Synthetics_Selenium:26
South America (São Paulo)	arn:aws:lambda:sa-east-1:783765544751:layer:Synthetics_Selenium:40
Amazon GovCloud (US-East)	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:Synthetics_Selenium:36

Region	ARN
Amazon GovCloud (US-West)	`arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:Synthetics_Selenium:37`

ARNs for syn-python-selenium-3.0

The following table lists the ARNs to use for version `syn-python-selenium-3.0` of the CloudWatch Synthetics runtime in each Amazon Region where it is available.

Region	ARN
US East (N. Virginia)	`arn:aws:lambda:us-east-1:378653112637:layer:Synthetics_Selenium:32`
US East (Ohio)	`arn:aws:lambda:us-east-2:772927465453:layer:Synthetics_Selenium:34`
US West (N. California)	`arn:aws:lambda:us-west-1:332033056316:layer:Synthetics_Selenium:32`
US West (Oregon)	`arn:aws:lambda:us-west-2:760325925879:layer:Synthetics_Selenium:34`
Africa (Cape Town)	`arn:aws:lambda:af-south-1:461844272066:layer:Synthetics_Selenium:32`
Asia Pacific (Hong Kong)	`arn:aws:lambda:ap-east-1:129828061636:layer:Synthetics_Selenium:32`
Asia Pacific (Hyderabad)	`arn:aws:lambda:ap-south-2:280298676434:layer:Synthetics_Selenium:20`
Asia Pacific (Jakarta)	`arn:aws:lambda:ap-southeast-3:246953257743:layer:Synthetics_Selenium:26`
Asia Pacific (Malaysia)	`arn:aws:lambda:ap-southeast-5:035872523913:layer:Synthetics_Selenium:7`

Region	ARN
Asia Pacific (Melbourne)	arn:aws:lambda:ap-southeast-4:200724813040:layer:Synthetics_Selenium:18
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:724929286329:layer:Synthetics_Selenium:32
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:608016332111:layer:Synthetics_Selenium:30
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:989515803484:layer:Synthetics_Selenium:34
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:068035103298:layer:Synthetics_Selenium:37
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:584677157514:layer:Synthetics_Selenium:32
Asia Pacific (Thailand)	arn:aws:lambda:ap-southeast-7:851725245975:layer:Synthetics_Selenium:3
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:172291836251:layer:Synthetics_Selenium:32
Canada (Central)	arn:aws:lambda:ca-central-1:236629016841:layer:Synthetics_Selenium:32
Canada West (Calgary)	arn:aws:lambda:ca-west-1:944448206667:layer:Synthetics_Selenium:76
China (Beijing)	arn:aws-cn:lambda:cn-north-1:422629156088:layer:Synthetics_Selenium:32
China (Ningxia);	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:Synthetics_Selenium:32

Region	ARN
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:122305336817:layer:Synthetics_Selenium:32
Europe (Ireland)	arn:aws:lambda:eu-west-1:563204233543:layer:Synthetics_Selenium:34
Europe (London)	arn:aws:lambda:eu-west-2:565831452869:layer:Synthetics_Selenium:32
Europe (Milan)	arn:aws:lambda:eu-south-1:525618516618:layer:Synthetics_Selenium:33
Europe (Paris)	arn:aws:lambda:eu-west-3:469466506258:layer:Synthetics_Selenium:32
Europe (Spain)	arn:aws:lambda:eu-south-2:029793053121:layer:Synthetics_Selenium:20
Europe (Stockholm)	arn:aws:lambda:eu-north-1:162938142733:layer:Synthetics_Selenium:32
Europe (Zurich)	arn:aws:lambda:eu-central-2:224218992030:layer:Synthetics_Selenium:19
Israel (Tel Aviv)	arn:aws:lambda:il-central-1:313249807427:layer:Synthetics_Selenium:17
Mexico (Central)	arn:aws:lambda:mx-central-1:654654265476:layer:Synthetics_Selenium:4
Middle East (Bahrain)	arn:aws:lambda:me-south-1:823195537320:layer:Synthetics_Selenium:32
Middle East (UAE)	arn:aws:lambda:me-central-1:239544149032:layer:Synthetics_Selenium:19

Region	ARN
South America (São Paulo)	`arn:aws:lambda:sa-east-1:783765544751:layer:Synthetics_Selenium:33`
Amazon GovCloud (US-East)	`arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:Synthetics_Selenium:30`
Amazon GovCloud (US-West)	`arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:Synthetics_Selenium:31`

ARNs for syn-python-selenium-2.1

The following table lists the ARNs to use for version `syn-python-selenium-2.1` of the CloudWatch Synthetics runtime in each Amazon Region where it is available.

Region	ARN
US East (N. Virginia)	`arn:aws:lambda:us-east-1:378653112637:layer:Synthetics:29`
US East (Ohio)	`arn:aws:lambda:us-east-2:772927465453:layer:Synthetics:31`
US West (N. California)	`arn:aws:lambda:us-west-1:332033056316:layer:Synthetics:29`
US West (Oregon)	`arn:aws:lambda:us-west-2:760325925879:layer:Synthetics:31`
Africa (Cape Town)	`arn:aws:lambda:af-south-1:461844272066:layer:Synthetics:29`
Asia Pacific (Hong Kong)	`arn:aws:lambda:ap-east-1:129828061636:layer:Synthetics:29`
Asia Pacific (Hyderabad)	`arn:aws:lambda:ap-south-2:280298676434:layer:Synthetics:17`

Region	ARN
Asia Pacific (Jakarta)	`arn:aws:lambda:ap-southeast-3:246953257743:layer:Synthetics:23`
Asia Pacific (Melbourne)	`arn:aws:lambda:ap-southeast-4:200724813040:layer:Synthetics:15`
Asia Pacific (Mumbai)	`arn:aws:lambda:ap-south-1:724929286329:layer:Synthetics:29`
Asia Pacific (Osaka)	`arn:aws:lambda:ap-northeast-3:608016332111:layer:Synthetics:27`
Asia Pacific (Seoul)	`arn:aws:lambda:ap-northeast-2:989515803484:layer:Synthetics:30`
Asia Pacific (Singapore)	`arn:aws:lambda:ap-southeast-1:068035103298:layer:Synthetics:34`
Asia Pacific (Sydney)	`arn:aws:lambda:ap-southeast-2:584677157514:layer:Synthetics:29`
Asia Pacific (Tokyo)	`arn:aws:lambda:ap-northeast-1:172291836251:layer:Synthetics:29`
Canada (Central)	`arn:aws:lambda:ca-central-1:236629016841:layer:Synthetics:29`
Canada West (Calgary)	`arn:aws:lambda:ca-west-1:944448206667:layer:Synthetics:73`
China (Beijing)	`arn:aws-cn:lambda:cn-north-1:422629156088:layer:Synthetics:29`
China (Ningxia);	`arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:Synthetics:29`

Region	ARN
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:122305336817:layer:Synthetics:29
Europe (Ireland)	arn:aws:lambda:eu-west-1:563204233543:layer:Synthetics:31
Europe (London)	arn:aws:lambda:eu-west-2:565831452869:layer:Synthetics:29
Europe (Milan)	arn:aws:lambda:eu-south-1:525618516618:layer:Synthetics:30
Europe (Paris)	arn:aws:lambda:eu-west-3:469466506258:layer:Synthetics:29
Europe (Spain)	arn:aws:lambda:eu-south-2:029793053121:layer:Synthetics:17
Europe (Stockholm)	arn:aws:lambda:eu-north-1:162938142733:layer:Synthetics:29
Europe (Zurich)	arn:aws:lambda:eu-central-2:224218992030:layer:Synthetics:16
Israel (Tel Aviv)	arn:aws:lambda:il-central-1:313249807427:layer:Synthetics:14
Middle East (Bahrain)	arn:aws:lambda:me-south-1:823195537320:layer:Synthetics:29
Middle East (UAE)	arn:aws:lambda:me-central-1:239544149032:layer:Synthetics:16
South America (São Paulo)	arn:aws:lambda:sa-east-1:783765544751:layer:Synthetics:30

Region	ARN
Amazon GovCloud (US-East)	`arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:Synthetics:29`
Amazon GovCloud (US-West)	`arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:Synthetics:29`

Common errors

Error: Running Amazon SAM projects locally requires Docker. Have you got it installed and running?

Make sure to start Docker on your computer.

SAM local invoke failed: An error occurred (ExpiredTokenException) when calling the GetLayerVersion operation: The security token included in the request is expired

Make sure that the Amazon default profile is set up.

More common errors

For more information about common errors with the SAM, see [Amazon SAM CLI troubleshooting](#).

Troubleshooting a failed canary

If your canary fails, check the following for troubleshooting.

General troubleshooting

- Use the canary details page to find more information. In the CloudWatch console, choose **Canaries** in the navigation pane and then choose the name of the canary to open the canary details page. In the **Availability** tab, check the **SuccessPercent** metric to see whether the problem is constant or intermittent.

  While still in the **Availability** tab, choose a failed data point to see screenshots, logs, and step reports (if available) for that failed run.

  If a step report is available because steps are part of your script, check to see which step has failed and see the associated screenshots to see the issue that your customers are seeing.

You can also check the HAR files to see if one or more requests are failing. You can dig deeper by using logs to drill down on failed requests and errors. Finally, you can compare these artifacts with the artifacts from a successful canary run to pinpoint the issue.

By default, CloudWatch Synthetics captures screenshots for each step in a UI canary. However, your script might be configured to disable screenshots. During debugging, you may want to enable screenshots again. Similarly, for API canaries you might want to see HTTP request and response headers and body during debugging. For information about how to include this data in the report, see executeHttpStep(stepName, requestOptions, [callback], [stepConfig]).

- If you had a recent deployment to your application, roll it back and then debug later.
- Connect to your endpoint manually to see if you can reproduce the same issue.

Topics

Canary fails after Lambda environment update

CloudWatch Synthetics canaries are implemented as Lambda functions in your account. These Lambda functions are subject to regular Lambda runtime updates containing security updates, bug fixes, and other improvements. Lambda strives to provide runtime updates that are backward-compatible with existing functions. However, as with software patching, there are rare cases in which a runtime update can negatively impact an existing function. If you believe your canary has been impacted by a Lambda runtime update, you can use the Lambda runtime management manual mode (in supported Regions) to temporarily roll back the Lambda runtime version. This keeps your canary function working and minimizes disruption, providing time to remedy the incompatibility before returning to the latest runtime version.

If your canary is failing after a Lambda runtime update, the best solution is to upgrade to one of the newest Synthetics runtimes. For more information about the latest runtimes, see Synthetics runtime versions.

As an alternative solution, in Regions where Lambda runtime management controls are available, you can revert a canary back to an older Lambda managed runtime, using manual mode for runtime management controls. You can set manual mode using either the Amazon CLI or by using the Lambda console, using the steps below in the following sections.

> **Warning**
>
> When you change the runtime settings to manual mode, your Lambda function won't receive automatic security updates until it is reverted back to Auto mode. During this period, your Lambda function might be susceptible to security vulnerabilities.

Prerequisites

- Install jq
- Install the latest version of the Amazon CLI. For more information, see Amazon CLI install and update instructions .

Step 1: Obtain the Lambda function ARN

Run the following command to retrieve the `EngineArn` field from the response. This `EngineArn` is the ARN of the Lambda function that is associated with the canary. You will use this ARN in the following steps.

```
aws synthetics get-canary --name my-canary | jq '.Canary.EngineArn'
```

Example output of `EngingArn`:

```
"arn:aws:lambda:us-west-2:123456789012:function:cwsyn-my-canary-dc5015c2-db17-4cb5-
afb1-EXAMPLE991:8"
```

Step 2: Obtain the last good Lambda runtime version ARN

To help understand whether your canary was impacted by a Lambda runtime update, check whether the date and time when the Lambda runtime version ARN changes in your logs appeared to the date and time when you saw impact to your canary. If they do not match, it is probably not a Lambda runtime update that is causing your issues.

If your canary is impacted by a Lambda runtime update, you must identify the ARN of the working Lambda runtime version that you were previously using. Follow the instructions in Identifying runtime version changes to find the ARN of the previous runtime. Record the runtime version ARN, and continue to Step 3. for setting the runtime management configuration.

If your canary has not yet been impacted by a Lambda environment update, then you can find the ARN of the Lambda runtime version that you are currently using. Run the following command to retrieve the `RuntimeVersionArn` of the Lambda function from the response.

```
aws lambda get-function-configuration \
--function-name "arn:aws:lambda:us-west-2:123456789012:function:cwsyn-my-canary-
dc5015c2-db17-4cb5-afb1-EXAMPLE991:8" | jq '.RuntimeVersionConfig.RuntimeVersionArn'
```

Example output of `RuntimeVersionArn`:

```
"arn:aws:lambda:us-
west-2::runtime:EXAMPLE647b82f490a45d7ddd96b557b916a30128d9dcab5f4972911ec0f"
```

Step 3: Updating the Lambda runtime management configuration

You can use either the Amazon CLI or the Lambda console to update the runtime management configuration.

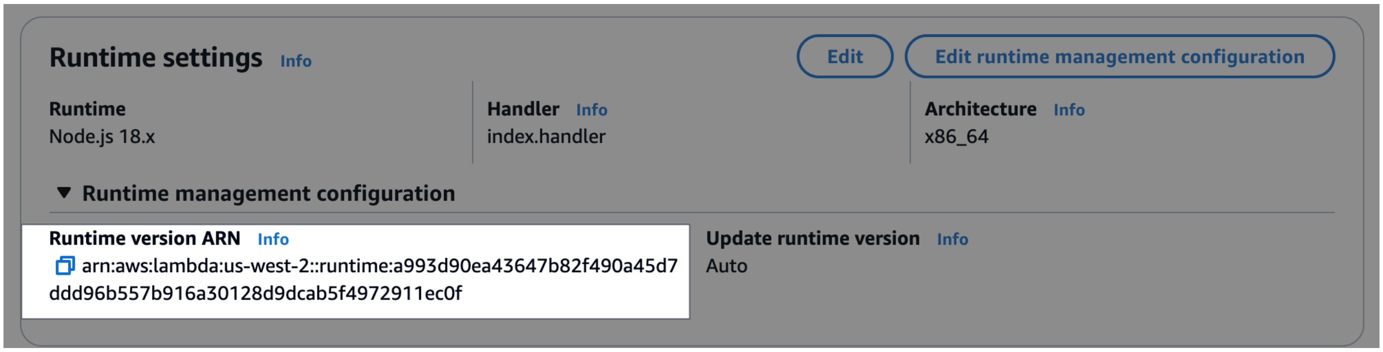
To set Lambda runtime management configuration manual mode using the Amazon CLI

Enter the following command to change the runtime management of the Lambda function to manual mode. Be sure to replace the *`function-name`* and *`qualifier`* with the Lambda function ARN and Lambda function version number respectively, using the values you found in Step 1. Also replace the the *`runtime-version-arn`* with the version ARN that you found in Step 2.

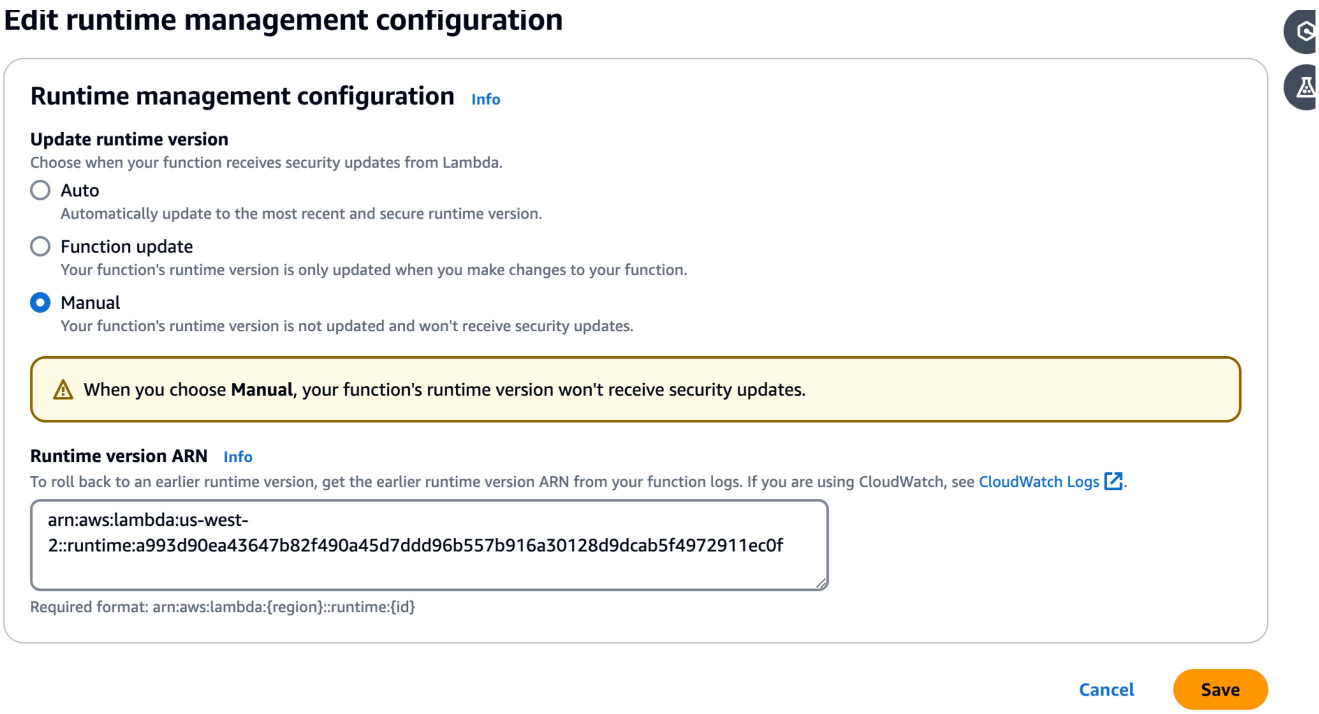
```
aws lambda put-runtime-management-config \
    --function-name "arn:aws:lambda:us-west-2:123456789012:function:cwsyn-my-canary-dc5015c2-db17-4cb5-afb1-EXAMPLE991" \
    --qualifier 8 \
    --update-runtime-on "Manual" \
    --runtime-version-arn "arn:aws:lambda:us-west-2::runtime:a993d90ea43647b82f490a45d7ddd96b557b916a30128d9dcab5f4972911ec0f"
```

To change a canary to manual mode using the Lambda console

1. Open the Amazon Lambda console at https://console.amazonaws.cn/lambda/.
2. Choose the **Versions** tab, choose the version number link that corresponds to your ARN, and choose the **Code** tab.
3. Scroll down to **Runtime settings**, expand **Runtime management configuration**, and copy the the **Runtime version ARN**.



4. Choose **Edit runtime management configuration**, choose **Manual**, paste the runtime version ARN that you copied earlier into the **Runtime version ARN** field. Then choose **Save**.



My canary is blocked by Amazon WAF

To allow canary traffic through Amazon WAF,create a Amazon WAF string match condition that allows a custom string that you specify. For more information, see Working with string match conditions in the Amazon WAF documentation.

We strongly recommend that you use your own custom user-agent string instead of using default values. This provides better control over Amazon WAF filtering and improves security.

To set a custom user-agent string, do the following:

- For Playwright runtimes, you can append your Amazon WAF approved custom user-agent string using the Synthetics configuration file. For more information, see CloudWatch Synthetics configurations.
- For Puppeteer or Selenium runtimes, you can add your custom user-agent string using supported library functions. For Puppeteer runtimes, see async addUserAgent(page, userAgentString);. For Selenium runtimes, see add_user_agent(user_agent_str).

Waiting for an element to appear

After analyzing your logs and screenshots, if you see that your script is waiting for an element to appear on screen and times out, check the relevant screenshot to see if the element appears on the page. Verify your `xpath` to make sure that it is correct.

For Puppetteer-related issues, check [Puppeteer's GitHub page](#) or internet forums.

Node is either not visible or not an HTMLElement for page.click()

If a node is not visible or is not an `HTMLElement` for `page.click()`, first verify the `xpath` that you are using to click the element. Also, if your element is at the bottom of the screen, adjust your viewport. CloudWatch Synthetics by default uses a viewport of 1920 * 1080. You can set a different viewport when you launch the browser or by using the Puppeteer function `page.setViewport`.

Unable to upload artifacts to S3, Exception: Unable to fetch S3 bucket location: Access Denied

If your canary fails because of an Amazon S3 error, CloudWatch Synthetics was unable to upload screenshots, logs, or reports created for the canary because of permission issues. Check the following:

- Check that the canary's IAM role has the `s3:ListAllMyBuckets` permission, the `s3:GetBucketLocation` permission for the correct Amazon S3 bucket, and the `s3:PutObject` permission for the bucket where the canary stores its artifacts. If the canary performs visual monitoring, the role also needs the `s3:GetObject` permission for the bucket. These same permissions are also required in the Amazon VPC S3 Gateway Endpoint Policy, if the canary is deployed in a VPC with a VPC endpoint.
- If the canary uses an Amazon KMS customer managed key for encryption instead of the standard Amazon managed key (default), the canary's IAM role might not have the permission to encrypt or decrypt using that key. For more information, see [Encrypting canary artifacts](#).
- Your bucket policy might not allow the encryption mechanism that the canary uses. For example, if your bucket policy mandates to use a specific encryption mechanism or KMS key, then you must select the same encryption mode for your canary.

If the canary performs visual monitoring, see [Updating artifact location and encryption when using visual monitoring](#) for more information.

Error: Protocol error (Runtime.callFunctionOn): Target closed.

This error appears if there are some network requests after the page or browser is closed. You might have forgotten to wait for an asynchronous operation. After executing your script, CloudWatch Synthetics closes the browser. The execution of any asynchronous operation after the browser is closed might cause `target closed error`.

Canary Failed. Error: No datapoint - Canary Shows timeout error

This means that your canary run exceeded the timeout. The canary execution stopped before CloudWatch Synthetics could publish success percent CloudWatch metrics or update artifacts such as HAR files, logs and screenshots. If your timeout is too low, you can increase it.

By default, a canary timeout value is equal to its frequency. You can manually adjust the timeout value to be less than or equal to the canary frequency. If your canary frequency is low, you must increase the frequency to increase the timeout. You can adjust both the frequency and the timeout value under **Schedule** when you create or update a canary by using the CloudWatch Synthetics console.

Be sure that your canary timeout value is no shorter than 15 seconds to allow for Lambda cold starts and the time it takes to boot up the canary instrumentation.

Canary artifacts are not available to view in the CloudWatch Synthetics console when this error happens. You can use CloudWatch Logs to see the canary's logs.

To use CloudWatch Logs to see the logs for a canary

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the left navigation pane, choose **Log groups**.
3. Find the log group by typing the canary name in the filter box. Log groups for canaries have the name **/aws/lambda/cwsyn-*canaryName*-randomId**.

Trying to access an internal endpoint

If you want your canary to access an endpoint on your internal network, we recommend that you set up CloudWatch Synthetics to use VPC. For more information, see [Running a canary on a VPC](#).

Canary runtime version upgrade and downgrade issues

If you recently upgraded the canary from runtime version `syn-1.0` to a later version, it may be a cross-origin request sharing (CORS) issue. For more information, see [Cross-origin request sharing (CORS) issue](#).

If you recently downgraded the canary to an older runtime version, check to make sure that the CloudWatch Synthetics functions that you are using are available in the older runtime version that you downgraded to. For example, the `executeHttpStep` function is available for runtime version `syn-nodejs-2.2` and later. To check on the availability of functions, see [Writing a canary script](#).

> **Note**
>
> When you plan to upgrade or downgrade the runtime version for a canary, we recommend that you first clone the canary and update the runtime version in the cloned canary. Once you have verified that the clone with the new runtime version works, you can update the runtime version of your original canary and delete the clone.

Cross-origin request sharing (CORS) issue

In a UI canary, if some network requests are failing with `403` or `net::ERR_FAILED`, check whether the canary has active tracing enabled and also uses the Puppeteer function `page.setExtraHTTPHeaders` to add headers. If so, the failed network requests might be caused by cross-origin request sharing (CORS) restrictions. You can confirm whether this is the case by disabling active tracing or removing the extra HTTP headers.

Why does this happen?

When active tracing is used, an extra header is added to all outgoing requests to trace the call. Modifying the request headers by adding a trace header or adding extra headers using Puppeteer's `page.setExtraHTTPHeaders` causes a CORS check for XMLHttpRequest (XHR) requests.

If you don't want to disable active tracing or remove the extra headers, you can update your web application to allow cross-origin access or you can disable web security by using the `disable-web-security` flag when you launch the Chrome browser in your script.

You can override launch parameters used by CloudWatch Synthetics and pass additional `disable-web-security` flag parameters by using the CloudWatch Synthetics launch function. For more information, see [Library functions available for Node.js canary scripts using Puppeteer](#).

> **Note**
>
> You can override launch parameters used by CloudWatch Synthetics when you use runtime version `syn-nodejs-2.1` or later.

Canary race condition issues

For the best experience when using CloudWatch Synthetics, ensure that the code written for the canaries is idempotent. Otherwise, in rare cases, canary runs may encounter race conditions when the canary interacts with the same resource across different runs.

Troubleshooting a canary on a VPC

If you have issues after creating or updating a canary on a VPC, one of the following sections might help you troubleshoot the problem.

New canary in error state or canary can't be updated

If you create a canary to run on a VPC and it immediately goes into an error state, or you can't update a canary to run on a VPC, the canary's role might not have the right permissions. To run on a VPC, a canary must have the permissions `ec2:CreateNetworkInterface`, `ec2:DescribeNetworkInterfaces`, and `ec2:DeleteNetworkInterface`. These permissions are all contained in the `AWSLambdaVPCAccessExecutionRole` managed policy. For more information, see Execution Role and User Permissions.

If this issue happened when you created a canary, you must delete the canary, and create a new one. If you use the CloudWatch console to create the new canary, under **Access Permissions**, select **Create a new role**. A new role that includes all permissions required to run the canary is created.

If this issue happens when you update a canary, you can update the canary again and provide a new role that has the required permissions.

"No test result returned" error

If a canary displays a "no test result returned" error, one of the following issues might be the cause:

- If your VPC does not have internet access, you must use VPC endpoints to give the canary access to CloudWatch and Amazon S3. You must enable the **DNS resolution** and **DNS hostname** options in the VPC for these endpoint addresses to resolve correctly. For more information, see

  Using DNS with Your VPC and Using CloudWatch and CloudWatch Synthetics with interface VPC endpoints .

- Canaries must run in private subnets within a VPC. To check this, open the **Subnets** page in the VPC console. Check the subnets that you selected when configuring the canary. If they have a path to an internet gateway (**igw-**), they are not private subnets.

To help you troubleshoot these issues, see the logs for the canary.

To see the log events from a canary

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Log groups**.
3. Choose the name of the canary's log group. The log group name starts with `/aws/lambda/cwsyn-canary-name`.

Troubleshooting an auto retry canary

To understand why your canary is failing or to analyze specific failed attempts, follow these troubleshooting steps.

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Application Signals**, **Synthetics Canaries**.
3. Choose the **Canary**.
4. Under the **Availability** tab, you can examine the run details by either:

   - Selecting a specific point on the Canary Runs graph
   - Under **Issues**, selecting a record. Note that retry attempts are tagged and share timestamps with their scheduled runs

   You can view additional information under **Steps**, **Screenshot**, **Logs**, **HAR file**, or **Traces (if active tracing is enabled)**.
5. Under **Canary artifacts and Amazon S3 location**, you can access the artifact and navigate to the Amazon S3 folders or buckets through the available links.
6. The **Canary runs** graph uses different colored points to indicate various status:

   - Blue Points – Indicates successful scheduled runs with a consistent value of 100%

- Red Points – Displays failure of both scheduled runs and all retries, marked at 0%
- Orange Points – Displays either 0% or 100%. 0% indicates ongoing retry following previous attempt failures and 100% means success was achieved after retrying

Sample code for canary scripts

This section contains code samples that illustrate some possible functions for CloudWatch Synthetics canary scripts.

Samples for Node.js and Playwright

Playwright canary with multiple steps

The following script is an example of a Node.js Playwright canary with multiple steps.

```
import { synthetics } from '@amzn/synthetics-playwright';

export async function handler(event, context) {
  try {
    console.log('Running Synthetics Playwright canary');
    const browser = await synthetics.launch();
    const browserContext = await browser.newContext();
    const page = await synthetics.getPage(browserContext);


    // Add steps
    // Step 1
    await synthetics.executeStep("home-page", async () => {
      console.log("Verify home page loads")
      await page.goto('https://www.amazon.com', {waitUntil: "load"});
      await new Promise(r => setTimeout(r, 5000));
    });

    // Step 2
    await synthetics.executeStep("search", async () => {
      console.log("Searching for a product")
      const searchInput = page.getByPlaceholder("Search Amazon").first();
      await searchInput.click()
      await searchInput.fill('Amazon echo');
      const btn = page.getByRole('button', { name: 'Go' }).first()
      await btn.click({ timeout: 15000 })
      console.log("Clicked search button")
```

```
    });

    // Step 3
    await synthetics.executeStep("search-results", async () => {
      console.log("Verifying search results")
      const resultsHeading = page.getByText("Results", {exact: true}).first()
      await resultsHeading.highlight();
      await new Promise(r => setTimeout(r, 5000));
    });

  } finally {
    // Close all browser contexts and browser
    await synthetics.close();
  }
}
```

Playwright canaries setting cookies

The following script is an example of a Node.js Playwright canary setting three cookies.

```
import { synthetics } from '@amzn/synthetics-playwright';

export const handler = async (event, context) => {
  try {
    let url = "http://smile.amazon.com/";
    const browser = await synthetics.launch();
    const page = await synthetics.getPage(browser);
    const cookies = [{
        'name': 'cookie1',
        'value': 'val1',
        'url': url
    },
    {
        'name': 'cookie2',
        'value': 'val2',
        'url': url
    },
    {
        'name': 'cookie3',
        'value': 'val3',
        'url': url
    }
   ];
   await page.context().addCookies(cookies);
```

```
    await page.goto(url, {waitUntil: 'load', timeout: 30000});
    await page.screenshot({ path: '/tmp/smile.png' });

  } finally {
    await synthetics.close();
  }
};
```

Samples for Node.js and Puppeteer

Setting cookies

Web sites rely on cookies to provide custom functionality or track users. By setting cookies in CloudWatch Synthetics scripts, you can mimic this custom behavior and validate it.

For example, a web site might display a **Login** link for a revisiting user instead of a **Register** link.

```
var synthetics = require('Synthetics');
const log = require('SyntheticsLogger');

const pageLoadBlueprint = async function () {

    let url = "http://smile.amazon.com/";

    let page = await synthetics.getPage();

    // Set cookies.  I found that name, value, and either url or domain are required
 fields.
    const cookies = [{
      'name': 'cookie1',
      'value': 'val1',
      'url': url
    },{
      'name': 'cookie2',
      'value': 'val2',
      'url': url
    },{
      'name': 'cookie3',
      'value': 'val3',
      'url': url
    }];

    await page.setCookie(...cookies);
```

```
    // Navigate to the url
    await synthetics.executeStep('pageLoaded_home', async function (timeoutInMillis =
 30000) {

        var response = await page.goto(url, {waitUntil: ['load', 'networkidle0'],
 timeout: timeoutInMillis});

        // Log cookies for this page and this url
        const cookiesSet = await page.cookies(url);
        log.info("Cookies for url: " + url + " are set to: " +
 JSON.stringify(cookiesSet));
    });

};

exports.handler = async () => {
    return await pageLoadBlueprint();
};
```

Device emulation

You can write scripts that emulate various devices so that you can approximate how a page looks and behaves on those devices.

The following sample emulates an iPhone 6 device. For more information about emulation, see page.emulate(options) in the Puppeteer documentation.

```
var synthetics = require('Synthetics');
const log = require('SyntheticsLogger');
const puppeteer = require('puppeteer-core');

const pageLoadBlueprint = async function () {

    const iPhone = puppeteer.devices['iPhone 6'];

    // INSERT URL here
    const URL = "https://amazon.com";

    let page = await synthetics.getPage();
    await page.emulate(iPhone);

    //You can customize the wait condition here. For instance,
```

```
    //using 'networkidle2' may be less restrictive.
    const response = await page.goto(URL, {waitUntil: 'domcontentloaded', timeout:
 30000});
    if (!response) {
        throw "Failed to load page!";
    }

    await page.waitFor(15000);

    await synthetics.takeScreenshot('loaded', 'loaded');

    //If the response status code is not a 2xx success code
    if (response.status() < 200 || response.status() > 299) {
        throw "Failed to load page!";
    }
};

exports.handler = async () => {
    return await pageLoadBlueprint();
};
```

Multi-step API canary

This sample code demonstrates an API canary with two HTTP steps: testing the same API for positive and negative test cases. The step configuration is passed to enable reporting of request/ response headers. Additionally, it hides the Authorization header and X-Amz-Security-Token, because they contain user credentials.

When this script is used as a canary, you can view details about each step and the associated HTTP requests such as step pass/fail, duration, and performance metrics like DNS look up time and first byte time. You can view the number of 2xx, 4xx and 5xx for your canary run.

```
var synthetics = require('Synthetics');
const log = require('SyntheticsLogger');

const apiCanaryBlueprint = async function () {

    // Handle validation for positive scenario
    const validatePositiveCase = async function(res) {
        return new Promise((resolve, reject) => {
            if (res.statusCode < 200 || res.statusCode > 299) {
                throw res.statusCode + ' ' + res.statusMessage;
```

```
            }

            let responseBody = '';
            res.on('data', (d) => {
                responseBody += d;
            });

            res.on('end', () => {
                // Add validation on 'responseBody' here if required. For ex, your
 status code is 200 but data might be empty
                resolve();
            });
        });
    };

    // Handle validation for negative scenario
    const validateNegativeCase = async function(res) {
        return new Promise((resolve, reject) => {
            if (res.statusCode < 400) {
                throw res.statusCode + ' ' + res.statusMessage;
            }

            resolve();
        });
    };

    let requestOptionsStep1 = {
        'hostname': 'myproductsEndpoint.com',
        'method': 'GET',
        'path': '/test/product/validProductName',
        'port': 443,
        'protocol': 'https:'
    };

    let headers = {};
    headers['User-Agent'] = [synthetics.getCanaryUserAgentString(), headers['User-
Agent']].join(' ');

    requestOptionsStep1['headers'] = headers;

    // By default headers, post data and response body are not included in the report
 for security reasons.
    // Change the configuration at global level or add as step configuration for
 individual steps
```

```
    let stepConfig = {
        includeRequestHeaders: true,
        includeResponseHeaders: true,
        restrictedHeaders: ['X-Amz-Security-Token', 'Authorization'], // Restricted
 header values do not appear in report generated.
        includeRequestBody: true,
        includeResponseBody: true
    };


    await synthetics.executeHttpStep('Verify GET products API with valid name',
 requestOptionsStep1, validatePositiveCase, stepConfig);

    let requestOptionsStep2 = {
        'hostname': 'myproductsEndpoint.com',
        'method': 'GET',
        'path': '/test/canary/InvalidName(',
        'port': 443,
        'protocol': 'https:'
    };

    headers = {};
    headers['User-Agent'] = [synthetics.getCanaryUserAgentString(), headers['User-
Agent']].join(' ');

    requestOptionsStep2['headers'] = headers;

    // By default headers, post data and response body are not included in the report
 for security reasons.
    // Change the configuration at global level or add as step configuration for
 individual steps
    stepConfig = {
        includeRequestHeaders: true,
        includeResponseHeaders: true,
        restrictedHeaders: ['X-Amz-Security-Token', 'Authorization'], // Restricted
 header values do not appear in report generated.
        includeRequestBody: true,
        includeResponseBody: true
    };

    await synthetics.executeHttpStep('Verify GET products API with invalid name',
 requestOptionsStep2, validateNegativeCase, stepConfig);

};
```

```
exports.handler = async () => {
    return await apiCanaryBlueprint();
};
```

Samples for Python and Selenium

The following sample Selenium code is a canary that fails with a custom error message when a target element is not loaded.

```
from aws_synthetics.selenium import synthetics_webdriver as webdriver
from aws_synthetics.common import synthetics_logger as logger
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.common.by import By

def custom_selenium_script():
    # create a browser instance
    browser = webdriver.Chrome()
    browser.get('https://www.example.com/')
    logger.info('navigated to home page')
    # set cookie
    browser.add_cookie({'name': 'foo', 'value': 'bar'})
    browser.get('https://www.example.com/')
    # save screenshot
    browser.save_screenshot('signed.png')
    # expected status of an element
    button_condition = EC.element_to_be_clickable((By.CSS_SELECTOR, '.submit-button'))
    # add custom error message on failure
    WebDriverWait(browser, 5).until(button_condition, message='Submit button failed to
 load').click()
    logger.info('Submit button loaded successfully')
    # browser will be quit automatically at the end of canary run,
    # quit action is not necessary in the canary script
    browser.quit()

# entry point for the canary
def handler(event, context):
    return custom_selenium_script()
```

Canaries and X-Ray tracing

You can choose to enable active Amazon X-Ray tracing on canaries that use the `syn-nodejs-2.0` or later runtime. With tracing enabled, traces are sent for all calls made by the canary that use the browser, the Amazon SDK, or HTTP or HTTPS modules. Canaries with tracing enabled appear on the X-Ray Trace Map, and within Application Signals after you have enabled it for your application.

> **Note**
>
> Activating X-Ray tracing on canaries is not yet supported in Asia Pacific (Jakarta).

When a canary appears on the X-Ray trace map, it appears as a new client node type. You can hover on a canary node to see data about latency, requests, and faults. You can also choose the canary node to see more data at the bottom of the page. From this area of the page, you can choose **View in Synthetics** to jump to the CloudWatch Synthetics console for more details about the canary, or choose **View Traces** to see more details about the traces from this canary's runs.

A canary with tracing enabled also has a **Tracing** tab in its details page, with details about traces and segments from the canary's runs.

Enabling tracing increases canary run time by 2.5% to 7%.

A canary with tracing enabled must use a role with the following permissions. If you use the console to create the role when you create the canary, it is given these permissions.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Sid230934",
            "Effect": "Allow",
            "Action": [
                "xray:PutTraceSegments"
            ],
            "Resource": "*"
        }
    ]
}
```

Traces generated by canaries incur charges. For more information about X-Ray pricing, see Amazon X-Ray Pricing.

Running a canary on a VPC

You can run canaries on endpoints on a VPC and public internal endpoints. To run a canary on a VPC, you must have both the **DNS Resolution** and **DNS hostnames** options enabled on the VPC. For more information, see Using DNS with Your VPC.

When you run a canary on a VPC endpoint, you must provide a way for it to send its metrics to CloudWatch and its artifacts to Amazon S3. If the VPC is already enabled for internet access, there's nothing more for you to do. The canary executes in your VPC, but can access the internet to upload its metrics and artifacts.

If the VPC is not already enabled for internet access, you have two options:

- Enable IPv4 internet access to allow the canary to send metrics to CloudWatch and Amazon S3. For more information, see the following section Giving internet access to your canary on a VPC.
- If you want to keep your VPC private, you can configure the canary to send its data to CloudWatch and Amazon S3 through private VPC endpoints. If you have not already done so, you must create a VPC endpoint for CloudWatch (com.amazonaws.*region*.monitoring) and a gateway endpoint for Amazon S3. For more information, see Using CloudWatch, CloudWatch Synthetics, and CloudWatch Network Monitoring with interface VPC endpoints and Amazon VPC Endpoints for Amazon S3.

Giving internet access to your canary on a VPC

Follow these steps to give internet access to your VPC canary, or to assign your canary a static IP address

To give internet access (IPv4) to a canary on a VPC

1. Create a NAT gateway in a public subnet on the VPC. For instructions, see Create a NAT gateway.
2. Add a new route to the route table in the private subnet where the canary is launched. Specify the following:

    - For **Destination**, enter **`0.0.0.0/0`**

- For **Target**, choose **NAT Gateway**, and then choose the ID of the NAT gateway that you created.
- Choose **Save routes**.

For more information about adding the route to the route table, see Add and remove routes from a route table.

To give internet access (IPv6) to a canary on a VPC

1. Configure your VPC to have Dualstack subnets. You must add an Egress-Only Internet Gateway to the VPC, update the route tables to allow traffic to the Internet Gateway, and allow outbound access from the associated Security Groups. For more information, see Add IPv6 support for your VPC .
2. Set the `Ipv6AllowedForDualstack` in your canary VPC configuration using the `CreateCanary` or `UpdateCanary` API. For more information, see VpcConfigInput.

   To enable outbound IPv6 traffic from your canary, the VPC subnets attached to the canary must be enabled for Dualstack.

> **Note**
>
> Be sure that the routes to your NAT gateway are in an **active** status. If the NAT gateway is deleted and you haven't updated the routes, they're in a black hole status. For more information, see Work with NAT gateways.

Encrypting canary artifacts

CloudWatch Synthetics stores canary artifacts such as screenshots, HAR files, and reports in your Amazon S3 bucket. By default, these artifacts are encrypted at rest using an Amazon managed key. For more information, see Customer keys and Amazon keys.

You can choose to use a different encryption option. CloudWatch Synthetics supports the following:

- **SSE-S3–** Server-side encryption (SSE) with an Amazon S3-managed key.

- **SSE-KMS**– Server-side encryption (SSE) with an Amazon KMS customer managed key.

If you want to use the default encryption option with an Amazon managed key, you don't need any additional permissions.

To use SSE-S3 encryption, you specify **SSE_S3** as the encryption mode when you create or update your canary. You do not need any additional permissions to use this encryption mode. For more information, see Protecting data using server-side encryption with Amazon S3-managed encryption keys (SSE-S3).

To use an Amazon KMS customer managed key, you specify **SSE-KMS** as the encryption mode when you create or update your canary, and you also provide the Amazon Resource Name (ARN) of your key. You can also use a cross-account KMS key.

To use a customer managed key, you need the following settings:

- The IAM role for your canary must have permission to encrypt your artifacts using your key. If you are using visual monitoring, you must also give it permission to decrypt artifacts.

```
{
    "Version": "2012-10-17",
    "Statement": {"Effect": "Allow",
        "Action": [
            "kms:GenerateDataKey",
            "kms:Decrypt"
        ],
        "Resource": "Your KMS key ARN"
    }
}
```

- Instead of adding permissions to your IAM role, you can add your IAM role to your key policy. If you use the same role for multiple canaries, you should consider this approach.

```
{
    "Sid": "Enable IAM User Permissions",
    "Effect": "Allow",
    "Principal": {
        "AWS": "Your synthetics IAM role ARN"
    },
    "Action": [
        "kms:GenerateDataKey",
```

```
        "kms:Decrypt"
    ],
    "Resource": "*"
}
```

- If you are using a cross-account KMS key, see Allowing users in other accounts to use a KMS key.

Viewing encrypted canary artifacts when using a customer managed key

To view canary artifacts, update your customer managed key to give Amazon KMS the decrypt permission to the user viewing the artifacts. Alternatively, add decrypt permissions to the user or IAM role that is viewing the artifacts.

The default Amazon KMS policy enables IAM policies in the account to allow access to the KMS keys. If you are using a cross-account KMS key, see Why are cross-account users getting Access Denied errors when they try to access Amazon S3 objects encrypted by a custom Amazon KMS key?.

For more information about troubleshooting access denied issues because of a KMS key, see Troubleshooting key access.

Updating artifact location and encryption when using visual monitoring

To perform visual monitoring, CloudWatch Synthetics compares your screenshots with baseline screenshots acquired in the run selected as the baseline. If you update your artifact location or encryption option, you must do one of the following:

- Ensure that your IAM role has sufficient permission for both the previous Amazon S3 location and the new Amazon S3 location for artifacts. Also ensure that it has permission for both the previous and new encryption methods and KMS keys.
- Create a new baseline by selecting the next canary run as a new baseline. If you use this option, you only need to ensure that your IAM role has sufficient permissions for the new artifact location and encryption option.

We recommend the second option of selecting the next run as the new baseline. This avoids having a dependency on an artifact location or encryption option that you're not using anymore for the canary.

For example, suppose that your canary uses artifact location A and KMS key K for uploading artifacts. If you update your canary to artifact location B and KMS key L, you can ensure that your

IAM role has permissions to both of the artifact locations (A and B) and both of the KMS keys (K and L). Alternatively, you can select the next run as the new baseline and ensure that your canary IAM role has permissions to artifact location B and KMS key L.

Viewing canary statistics and details

You can view details about your canaries and see statistics about their runs.

To be able to see all the details about your canary run results, you must be logged on to an account that has sufficient permissions. For more information, see Required roles and permissions for CloudWatch canaries.

To view canary statistics and details

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Application Signals**, **Synthetics Canaries**.

   In the details about the canaries that you have created:

   - **Status** visually shows how many of your canaries have passed their most recent runs.
   - **Groups** displays the groups you have created, and displays how many of them have failing or alarming canaries.
   - **Slowest performers** displays the group and the Region with the slowest-performing canaries. These are calculated by adding up the average duration of all canaries (across the time span selected) within a group or Region and dividing it by the number of canaries in the group or Region. If you choose the metric for Slowest group, the table is filtered to display only the slowest groups and their canaries. The table is sorted by Average Duration.
   - Near the bottom of the page is a table displaying all canaries. You can use the filtering bar to filter the table to show canaries by specific canary names, last run results, success percentage, alarms, run rates, canary state, runtimes, and unique tags.

     For the alarms column, only alarms that conform to the naming standard for canary alarms are displayed. This standard is `Synthetics-Alarm-`*`canaryName`*`-`*`index`*. Canary alarms that you create in the **Synthetics** section of the CloudWatch console automatically use this naming convention. If you create canary alarms in the **Alarms** section of the CloudWatch console or by using Amazon CloudFormation, and you don't use this naming convention, the alarms work but they do not appear in this list.

3. To see more details about a single canary, choose the name of the canary in the **Canaries** table.

   In the details about that canary:

   - The **Availability** tab displays information about the recent runs of this canary.

     Under **Canary runs**, you can choose one of the lines to see details about that run.

     Under the graph, you can choose **Steps**, **Screenshot**, **Logs**, or **HAR file** to see these types of details. If the canary has active tracing enabled, you can also choose **Traces** to see tracing information from the canary's runs.

     The logs for canary runs are stored in S3 buckets and in CloudWatch Logs.

     Screenshots show how your customers view your webpages. You can use the HAR files (HTTP Archive files) to view detailed performance data about the webpages. You can analyze the list of web requests and catch performance issues such as time to load for an item. Log files show the record of interactions between the canary run and the webpage and can be used to identify details of errors.

     If the canary uses the `syn-nodejs-2.0-beta` runtime or later, you can sort the HAR files by status code, request size, or duration.

     The **Steps** tab displays a list of the canary's steps, each step's status, failure reason, URL after step execution, screenshots, and duration of step execution. For API canaries with HTTP steps, you can view steps and corresponding HTTP requests if you are using runtime `syn-nodejs-2.2` or later.

     Choose the **HTTP Requests** tab to view the log of each HTTP request made by the canary. You can view request/response headers, response body, status code, error and performance timings (total duration, TCP connection time, TLS handshake time, first byte time, and content transfer time). All HTTP requests which use the HTTP/HTTPS module under the hood are captured here.

     By default in API canaries, the request header, response header, request body, and response body are not included in the report for security reasons. If you choose to include them, the data is stored only in your S3 bucket. For information about how to include this data in the report, see executeHttpStep(stepName, requestOptions, [callback], [stepConfig]).

Response body content types of text, HTML and JSON are supported. Content types like text/HTML, text/plain, application/JSON and application/x-amz-json-1.0 are supported. Compressed responses are not supported.

- The **Monitoring** tab displays graphs of the CloudWatch metrics published by this canary. For more information about these metrics, see CloudWatch metrics published by canaries.

  Below the CloudWatch graphics published by the canary are graphs of Lambda metrics related to the canary's Lambda code.
- The **Configuration** tab displays configuration and schedule information about the canary.
- The **Groups** tab displays the groups that this canary is associated with, if any.
- The **Tags** tab displays the tags associated with the canary.

CloudWatch metrics published by canaries

Canaries publish the following metrics to CloudWatch in the `CloudWatchSynthetics` namespace. For more information about viewing CloudWatch metrics, see View available metrics.

Metric	Description
`2xx`	The number of network requests performed by the canary that returned OK responses, with response codes between 200 and 299. This metric is reported for UI canaries that use runtime version `syn-nodejs-2.0` or later, and is reported for API canaries that use runtime version `syn-nodejs-2.2` or later. Valid Dimensions: CanaryName Valid Statistic: Sum Units: Count
`4xx`	The number of network requests performed by the canary that returned Error responses, with response codes between 400 and 499.

Metric	Description
	This metric is reported for UI canaries that use runtime version `syn-nodejs-2.0` or later, and is reported for API canaries that use runtime version `syn-nodejs-2.2` or later. Valid Dimensions: CanaryName Valid Statistic: Sum Units: Count
`5xx`	The number of network requests performed by the canary that returned Fault responses, with response codes between 500 and 599. This metric is reported for UI canaries that use runtime version `syn-nodejs-2.0` or later, and is reported for API canaries that use runtime version `syn-nodejs-2.2` or later. Valid Dimensions: CanaryName Valid Statistic: Sum Units: Count
`Duration`	The duration in milliseconds of the canary run. Valid Dimensions: CanaryName Valid Statistic: Average Units: Milliseconds
`Error`	The number of times the canary failed to run its full script. Valid Dimensions: CanaryName Valid Statistic: Sum

Metric	Description
`Failed`	The number of canary runs that failed to execute. These failures are related to the canary itself. Valid Dimensions: CanaryName Valid Statistic: Sum Units: Count
`Failed requests`	The number of HTTP requests executed by the canary on the target website that failed with no response. Valid Dimensions: CanaryName Valid Statistic: Sum Units: Count
`RetryCount`	The number of times your canary retried. This metric is only displayed when there are retries. Valid Dimensions: CanaryName Valid Statistic: Sum Units: Count
`SuccessPercent`	The percentage of the runs of this canary that succeed and find no failures. Valid Dimensions: CanaryName Valid Statistic: Average Units: Percent

Metric	Description
`SuccessPercentWithRetries`	The percentage of the runs of this canary that succeed after all attempts. Valid Dimensions: CanaryName Valid Statistic: Average Units: Percent
`VisualMonitoringSuccessPercent`	The percentage of visual comparisons that successfully matched the baseline screenshots during a canary run. Valid Dimensions: CanaryName Valid Statistic: Average Units: Percent
`VisualMonitoringTotalComparisons`	The total number of visual comparisons that happened during a canary run. Valid Dimensions: CanaryName Units: Count

> **Note**
>
> Canaries that use either the `executeStep()` or `executeHttpStep()` methods from the Synthetics library also publish `SuccessPercent` and `Duration` metrics with the dimensions `CanaryName` and `StepName` for each step.

Edit or delete a canary

You can edit or delete an existing canary.

Edit canary

When you edit a canary, even if you don't change its schedule, the schedule is reset corresponding to when you edit the canary. For example, if you have a canary that runs every hour, and you edit that canary, the canary will run immediately after the edit is completed and then every hour after that.

To edit or update a canary

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Application Signals**, **Synthetics Canaries**.
3. Select the button next to the canary name, and choose **Actions**, **Edit**.
4. (Optional) If this canary performs visual monitoring of screenshots and you want to set the next run of the canary as the baseline, select **Set next run as new baseline**.
5. (Optional) If this canary performs visual monitoring of screenshots and you want to remove a screenshot from visual monitoring or you want to designate parts of the screenshot to be ignored during visual comparisons, under **Visual Monitoring** choose **Edit Baseline**.

   The screenshot appears, and you can do one of the following:

   - To remove the screenshot from being used for visual monitoring, select **Remove screenshot from visual test baseline**.
   - To designate parts of the screenshot to be ignored during visual comparisons, click and drag to draw areas of the screen to ignore. Once you have done this for all the areas that you want to ignore during comparisons, choose **Save**.
6. Make any other changes to the canary that you'd like, and choose **Save**.

Delete canary

When you delete a canary, you can choose whether to also delete other resources used and created by the canary. If the canary’s `ProvisionedResourceCleanup` field is set to `AUTOMATIC` or `DeleteLambda` is specified as `true` when you delete the canary, CloudWatch Synthetics will automatically delete the Lambda functions and layers that are used by the canary.

When you delete a canary, you should also delete the following:

- Lambda functions and layers used by this canary. Their prefix is `cwsyn-MyCanaryName`.

- CloudWatch alarms created for this canary. These alarms have a name that starts with `Synthetics-Alarm-MyCanaryName`. For more information about deleting alarms, see [Edit or delete a CloudWatch alarm](#).
- Amazon S3 objects and buckets, such as the canary's results location and artifact location.
- IAM roles created for the canary. These have the name `role/service-role/CloudWatchSyntheticsRole-MyCanaryName`.
- Log groups in CloudWatch Logs created for the canary. These logs groups have the following names: `/aws/lambda/cwsyn-MyCanaryName-randomId`.

Before you delete a canary, you might want to view the canary details and make note of this information. That way, you can delete the correct resources after you delete the canary.

To delete a canary

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Application Signals**, **Synthetics Canaries**.
3. If the canary is currently in the `RUNNING` state, you must stop it. Only canaries in the `STOPPED`, `READY(NOT_STARTED)`, or `ERROR` states can be deleted.

   To stop the canary, select the button next to the canary name, and choose **Actions**, **Stop**.
4. Select the button next to the canary name, and choose **Actions**, **Delete**.
5. Choose whether to also delete the other resources created for and used by the canary. Lambda functions and layers will be deleted alongside the canary, but you can additionally choose to delete the canary's IAM role and IAM policy.

   Enter **`Delete`** into the box and choose **Delete**.
6. Delete the other resources used by and created for the canary, as listed earlier in this section.

Start, stop, delete, or update runtime for multiple canaries

You can stop, start, delete, or update the runtime of as many as five canaries with one action. If you update the runtime of a canary, it is updated to the latest runtime available for the language and framework that the canary uses.

If you select multiple canaries and only some of them are in a state that is valid for the action that you select, the action is performed only on the canaries where that action is valid. For example, if you select some canaries that are currently running and some that are not, and you select to start the canaries, the canaries that weren't already running will start, and the canaries that were already running are not affected.

If none of the canaries that you select are valid for an action, that action will not be available in the menu.

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Application Signals**, **Synthetics Canaries**.
3. Select the check boxes next to the canaries that you want to stop, start, or delete.
4. Choose **Actions** and then choose either **Start**, **Stop**, **Delete**, **Start Dry Run**, or **Update Runtime**.

   In addition, when choosing **Update Runtime**, you can choose to dry run the runtime update first before committing the change.

Monitoring canary events with Amazon EventBridge

Amazon EventBridge event rules can notify you when canaries change status or complete runs. EventBridge delivers a near-real-time stream of system events that describe changes in Amazon resources. CloudWatch Synthetics sends these events to EventBridge on a *best effort* basis. Best effort delivery means that CloudWatch Synthetics attempts to send all events to EventBridge, but in some rare cases an event might not be delivered. EventBridge processes all received events at least once. Additionally, your event listeners might not receive the events in the order that the events occurred.

> **Note**
>
> Amazon EventBridge is an event bus service that you can use to connect your applications with data from a variety of sources. For more information, see What is Amazon EventBridge? in the *Amazon EventBridge User Guide*.

CloudWatch Synthetics emits an event when a canary changes state or completes a run. You can create an EventBridge rule that includes an event pattern to match all event types sent from CloudWatch Synthetics, or that matches only specific event types. When a canary triggers a rule, EventBridge invokes the target actions defined in the rule. This allows you to send notifications, capture event information, and take corrective action, in response to a canary state change or the completion of a canary run. For example, you can create rules for the following use cases:

- Investigating when a canary run fails
- Investigating when a canary has gone into the ERROR state
- Tracking a canary's life cycle
- Monitoring canary run success or failure as part of a workflow

Example events from CloudWatch Synthetics

This section lists example events from CloudWatch Synthetics. For more information about event format, see Events and Event Patterns in EventBridge.

Canary status change

In this event type, the values of `current-state` and `previous-state` can be the following:

CREATING | READY | STARTING | RUNNING | UPDATING | STOPPING | STOPPED | ERROR

```
{
                "version": "0",
                "id": "8a99ca10-1e97-2302-2d64-316c5dedfd61",
                "detail-type": "Synthetics Canary Status Change",
                "source": "aws.synthetics",
                "account": "123456789012",
                "time": "2021-02-09T22:19:43Z",
                "region": "us-east-1",
                "resources": [],
                "detail": {
                                "account-id": "123456789012",
                                "canary-id": "EXAMPLE-dc5a-4f5f-96d1-989b75a94226",
                                "canary-name": "events-bb-1",
                                "current-state": "STOPPED",
                                "previous-state": "UPDATING",
                                "source-location": "NULL",
                                "updated-on": 1612909161.767,
                                "changed-config": {
                                                "executionArn": {
                                                                "previous-value":
 "arn:aws:lambda:us-east-1:123456789012:function:cwsyn-events-bb-1-af3e3a05-
dc5a-4f5f-96d1-989EXAMPLE:1",
```

```
                                                                "current-value":
 "arn:aws:lambda:us-east-1:123456789012:function:cwsyn-events-bb-1-af3e3a05-
dc5a-4f5f-96d1-989EXAMPLE:2"
                                                },
                                                "vpcId": {
                                                                "current-value": "NULL"
                                                },
                                                "testCodeLayerVersionArn": {
                                                                "previous-
value": "arn:aws:lambda:us-east-1:123456789012:layer:cwsyn-events-bb-1-af3e3a05-
dc5a-4f5f-96d1-989EXAMPLE:1",
                                                                "current-value":
 "arn:aws:lambda:us-east-1:123456789012:layer:cwsyn-events-bb-1-af3e3a05-
dc5a-4f5f-96d1-989EXAMPLE:2"
                                                }
                                },
                                "message": "Canary status has changed"
                }
}
```

Successful canary run completed

```
{
                "version": "0",
                "id": "989EXAMPLE-f4a5-57a7-1a8f-d9cc768a1375",
                "detail-type": "Synthetics Canary TestRun Successful",
                "source": "aws.synthetics",
                "account": "123456789012",
                "time": "2021-02-09T22:24:01Z",
                "region": "us-east-1",
                "resources": [],
                "detail": {
                                "account-id": "123456789012",
                                "canary-id": "989EXAMPLE-dc5a-4f5f-96d1-989b75a94226",
                                "canary-name": "events-bb-1",
                                "canary-run-id": "c6c39152-8f4a-471c-9810-989EXAMPLE",
                                "artifact-location": "cw-syn-results-123456789012-us-
east-1/canary/us-east-1/events-bb-1-ec3-28ddbe266797/2021/02/09/22/23-41-200",
                                "test-run-status": "PASSED",
                                "state-reason": "null",
                                "canary-run-timeline": {
                                                "started": 1612909421,
                                                "completed": 1612909441
```

```
                                },
                                "message": "Test run result is generated successfully"
                }
}
```

Failed canary run completed

```
{
                "version": "0",
                "id": "2644b18f-3e67-5ebf-cdfd-bf9f91392f41",
                "detail-type": "Synthetics Canary TestRun Failure",
                "source": "aws.synthetics",
                "account": "123456789012",
                "time": "2021-02-09T22:24:27Z",
                "region": "us-east-1",
                "resources": [],
                "detail": {
                                "account-id": "123456789012",
                                "canary-id": "af3e3a05-dc5a-4f5f-96d1-9989EXAMPLE",
                                "canary-name": "events-bb-1",
                                "canary-run-id": "0df3823e-7e33-4da1-8194-
b04e4d4a2bf6",
                                "artifact-location": "cw-syn-results-123456789012-us-
east-1/canary/us-east-1/events-bb-1-ec3-989EXAMPLE/2021/02/09/22/24-21-275",
                                "test-run-status": "FAILED",
                                "state-reason": "\"Error: net::ERR_NAME_NOT_RESOLVED
 \""
                                "canary-run-timeline": {
                                                "started": 1612909461,
                                                "completed": 1612909467
                                },
                                "message": "Test run result is generated successfully"
                }
}
```

It's possible that events might be duplicated or out of order. To determine the order of events, use the `time` property.

Prerequisites for creating EventBridge rules

Before you create an EventBridge rule for CloudWatch Synthetics, you should do the following:

- Familiarize yourself with events, rules, and targets in EventBridge.

- Create and configure the targets invoked by your EventBridge rules. Rules can invoke many types of targets, including:
  - Amazon SNS topics
  - Amazon Lambda functions
  - Kinesis streams
  - Amazon SQS queues

For more information, see What is Amazon EventBridge? and Getting started with Amazon EventBridge in the *Amazon EventBridge User Guide*.

Create an EventBridge rule (CLI)

The steps in the following example create an EventBridge rule that publishes an Amazon SNS topic when the canary named `my-canary-name` in `us-east-1` completes a run or changes state.

1. Create the rule.

```
aws events put-rule \
  --name TestRule \
  --region us-east-1 \
  --event-pattern "{\"source\": [\"aws.synthetics\"], \"detail\": {\"canary-name\": [\"my-canary-name\"]}}"
```

   Any properties you omit from the pattern are ignored.

2. Add the topic as a rule target.

   - Replace *topic-arn* with the Amazon Resource Name (ARN) of your Amazon SNS topic.

```
aws events put-targets \
  --rule TestRule \
  --targets "Id"="1","Arn"="topic-arn"
```

> **Note**
>
> To allow Amazon EventBridge to call your target topic, you must add a resource-based policy to your topic. For more information, see Amazon SNS permissions in the *Amazon EventBridge User Guide*.

For more information, see Events and event patterns in EventBridge in the *Amazon EventBridge User Guide*.

Performing safe canary updates

CloudWatch synthetics safe canary updates allows you to test the updates on your existing canaries before applying the changes. This feature helps you validate canary compatibility with new run times and other configuration changes such as code or memory changes. This will help minimize potential monitoring disruptions caused by erroneous updates.

By using canary safe updates on runtime version updates, configuration changes, and code script modifications, you can mitigate risk, maintain uninterrupted monitoring, verify the changes before committing, update, and reduce downtime.

Topics

- Prerequisites
- Best practices
- Testing canary using dry run
- Limitations

Prerequisites

Make sure the prequisites are complete.

- Amazon account with CloudWatch synthetics permissions
- Existing canary on the supported runtime versions (see Limitations for compatible runtimes)
- Include compatible runtimes when performing a dry run (see Limitations for compatible runtimes)

Best practices

Here are some best practices to follow while performing a canary .

- Execute a dry run to validate a runtime update
- Perform dry runs before production updates to canary
- Review canary logs and artifacts after a dry run
- Use dry runs to validate dependencies and library compatibility

Testing canary using dry run

You can test the canary update using the following options:

Using the Amazon Web Services Management Console's Edit workflow

1. Go the CloudWatch synthetics console.
2. Select the canary you want to update.
3. From the **Actions** drop down, choose **Edit**.

   Update the canary with the changes you want to test. For example, changing runtime version or editing the script’s code.
4. Under **Canary script**, choose **Start Dry Run** to test and view the results immediately or choose **Validate and save later** at the bottom of the page to start the test and view the results later in your **Canary Details** page.
5. After the dry run succeeds, choose **Submit** to commit your canary updates.

Using the Amazon Web Services Management Console for updating canaries in a batch

1. Go the CloudWatch synthetics console.
2. Choose the **Synthetics** list page.
3. Select upto five canaries for which you want to update the runtime.
4. From the **Actions** drop down, choose **Update Runtime**.
5. Choose **Start dry run for new runtime** to start the dry run and test your changes before an update.

6. On the **Synthetics** list page, you will see a text next to the **Runtime** version for the canary that displays the progress of the dry run (this is only displayed for dry runs involving a runtime update).

   Once the dry run succeeds, you will see an **Initiate Update** text.
7. Choose **Initiate Update** to commit the runtime update.
8. If the dry run fails, you will see an **Update dry run failed** text. Choose the text to view the debug link to the canary details page.

Using the Amazon CLI or SDK

The API starts the dry run for the provided canary name `MyCanary` and updates the runtime version to `syn-nodejs-puppeteer-10.0`.

```
aws synthetics start-canary-dry-run \
    --name MyCanary \
    --runtime-version syn-nodejs-puppeteer-10.0

      // Or if you wanted to update other configurations:

aws synthetics start-canary-dry-run \
    --name MyCanary \
    --execution-role-arn arn:aws:iam::123456789012:role/NewRole
```

The API will return the `DryRunId` inside the `DryRunConfigOutput`.

Call `GetCanary` with the provided `DryRunId` to receive the canary's dry run configurations and an additional field `DryRunConfig` which contains the status of the dry run listed as `LastDryRunExecutionStatus`.

```
aws synthetics get-canary \
    --name MyCanary \
    --dry-run-id XXXX-XXXX-XXXX-XXXX
```

For more details, use `GetCanaryRuns` with the provided `DryRunId` to retrieve the run and additional information.

```
aws synthetics get-canary-runs \
    --name MyCanary \
```

```
    --dry-run-id XXXX-XXXX-XXXX-XXXX
```

After a successful dry run, you can then use `UpdateCanary` with the provided `DryRunId` in order to commit your changes.

```
aws synthetics update-canary \
    --name MyCanary \
    --dry-run-id XXXX-XXXX-XXXX-XXXX
```

When it fails for any reason (result from GetCanaryRuns will have the details), the result from `GetCanaryRuns` has an artifact location that contains logs to debug. When there are no logs, the dry run failed to be created. You can validate by using `GetCanary`.

```
aws synthetics get-canary \
    --name MyCanary \
    --dry-run-id XXXX-XXXX-XXXX-XXXX
```

The *State*, *StateReason*, and *StateReasonCode* displays the status of the dry run.

Using Amazon CloudFormation

In your template for a Synthetics Canary, provide the field `DryRunAndUpdate` which accepts a boolean value `true` or `false`.

when the value is `true` every update executes a dry run to validate the changes before automatically updating the canary. When the dry run fails, the canary does not update and fails the deployment and Amazon CloudFormation deployment with a valid reason. To debug this issue, use the Amazon Synthetics console or if using an API, get the `ArtifactS3Location` using the `GetCanaryRuns` API, and download the `*-log.txt` files to review the canary log executions for errors. After validation, modify the Amazon CloudFormation template and retry the deployment or use the above API to validate.

When the value is `false`, synthetics will not execute a dry run to validate changes and will directly commit your updates.

For information on troubleshooting a failed canary, see Troubleshooting a failed canary.

An example template.

```
SyntheticsCanary:
    Type: 'AWS::Synthetics::Canary'
```

```
    Properties:
      Name: MyCanary
      RuntimeVersion: syn-nodejs-puppeteer-10.0
      Schedule: {Expression: 'rate(5 minutes)', DurationInSeconds: 3600}
      ...
      DryRunAndUpdate: true
```

Limitations

- Supports runtime versions – syn-nodejs-puppeteer-10.0+, syn-nodejs-playwright-2.0+, and syn-python-selenium-5.1+
- You can only execute one dry run per canary at a time
- When a dry run fails, you cannot update the canary
- Dry run cannot test any **Schedule** field changes

> **Note**
>
> When you initiate a dry run with code changes for a Playwright canary and you want to update the canary without providing the associated `DryRunId`, you must explicitly specify the code parameters.

CloudWatch RUM

With CloudWatch RUM, you can perform real user monitoring to collect and view client-side data about your web application performance from actual user sessions in near real time. The data that you can visualize and analyze includes page load times, client-side errors, and user behavior. When you view this data, you can see it all aggregated together and also see breakdowns by the browsers and devices that your customers use.

You can use the collected data to quickly identify and debug client-side performance issues. CloudWatch RUM helps you visualize anomalies in your application performance and find relevant debugging data such as error messages, stack traces, and user sessions. You can also use RUM to understand the range of end user impact including the number of users, geolocations, and browsers used.

End user data that you collect for CloudWatch RUM is retained for 30 days and then automatically deleted. If you want to keep the RUM events for a longer time, you can choose to have the app monitor send copies of the events to CloudWatch Logs in your account. Then, you can adjust the retention period for that log group.

To use RUM, you create an *app monitor* and provide some information. RUM generates a JavaScript snippet for you to paste into your application. The snippet pulls in the RUM web client code. The RUM web client captures data from a percentage of your application's user sessions, which is displayed in a pre-built dashboard. You can specify what percentage of user sessions to gather data from.

CloudWatch RUM is integrated with Application Signals, which can discover and monitor your application services, clients, Synthetics canaries, and service dependencies. Use Application Signals to see a list or visual map of your services, view health metrics based on your service level objectives (SLOs), and drill down to see correlated X-Ray traces for more detailed troubleshooting. To see RUM client page requests in Application Signals, turn on X-Ray active tracing by creating an app monitor, or manually configuring the RUM web client. Your RUM clients are displayed on the Service Map connected to your services, and in the Service detail page of the services they call.

The RUM web client is open source. For more information, see CloudWatch RUM web client.

Performance considerations

This section discusses the performance considerations of using CloudWatch RUM.

- **Load performance impact**— The CloudWatch RUM web client can be installed in your web application as a JavaScript module, or loaded into your web application asynchronously from a content delivery network (CDN). It does not block the application's load process. CloudWatch RUM is designed to have no perceptible impact on application load time.
- **Runtime impact**— The RUM web client performs processing to record and dispatch RUM data to the CloudWatch RUM service. Because events are infrequent and the amount of processing is small, CloudWatch RUM is designed for there to be no detectable impact to the application's performance.
- **Network impact**— The RUM web client periodically sends data to the CloudWatch RUM service. Data is dispatched at regular intervals while the application is running, and also immediately before the browser unloads the application. Data sent immediately before the browser unloads the application are sent as beacons, which, are designed to have no detectable impact on the application's unload time.

RUM Pricing

With CloudWatch RUM, you incur charges for every RUM event that CloudWatch RUM receives. Each data item collected using the RUM web client is considered a RUM event. Examples of RUM events include a page view, a JavaScript error, and an HTTP error. You have options for which types of events are collected by each app monitor. You can activate or deactivate options to collect performance telemetry events, JavaScript errors, HTTP errors, and X-Ray traces. For more information about choosing these options, see Creating a CloudWatch RUM app monitor and Information collected by the CloudWatch RUM web client. For more information about pricing, see Amazon CloudWatch Pricing.

Region availability

CloudWatch RUM is currently available in the following Regions:

- US East (N. Virginia)
- US East (Ohio)
- US West (N. California)
- US West (Oregon)
- Africa (Cape Town)
- Asia Pacific (Jakarta)
- Asia Pacific (Mumbai)
- Asia Pacific (Hyderabad)
- Asia Pacific (Melbourne)
- Asia Pacific (Osaka)
- Asia Pacific (Seoul)
- Asia Pacific (Singapore)
- Asia Pacific (Sydney)
- Asia Pacific (Tokyo)
- Asia Pacific (Hong Kong)
- Canada (Central)
- Europe (Frankfurt)
- Europe (Ireland)
- Europe (London)
- Europe (Milan)
- Europe (Paris)

- Europe (Spain)
- Europe (Stockholm)
- Europe (Zurich)
- Middle East (Bahrain)
- Middle East (UAE)
- South America (São Paulo)
- Israel (Tel Aviv)

Topics

IAM policies to use CloudWatch RUM

To be able to fully manage CloudWatch RUM, you must be signed in as an IAM user or role that has the **AmazonCloudWatchRUMFullAccess** IAM policy. Additionally, you may need other policies or permissions:

- To create an app monitor that creates a new Amazon Cognito identity pool for authorization, you need to have the **Admin** IAM role or the **AdministratorAccess** IAM policy.
- To create an app monitor that sends data to CloudWatch Logs, you must be logged on to an IAM role or policy that has the following permission:

```
{
    "Effect": "Allow",
    "Action": [
        "logs:PutResourcePolicy"
    ],
    "Resource": [
        "*"
    ]
}
```

- To enable JavaScript source maps in an app monitor, you will need to upload your source map files to a Amazon S3 bucket. Your IAM role or policy needs specific Amazon S3 permissions that allow creating Amazon S3 buckets, setting bucket policies, and managing files in the bucket. For security, scope these permissions to specific resources. The example policy below restricts access to buckets containing `rum` in their names and uses the `aws:ResourceAccount` condition key to limit permissions to the principal account only.

```
{
    "Sid": "AllowS3BucketCreationAndListing",
    "Effect": "Allow",
    "Action": [
        "s3:CreateBucket",
        "s3:ListAllMyBuckets"
    ],
    "Resource": "arn:aws:s3:::*",
    "Condition": {
        "StringEquals": {
            "aws:ResourceAccount": "${aws:PrincipalAccount}"
        }
    }
},
{
    "Sid": "AllowS3BucketActions",
    "Effect": "Allow",
    "Action": [
        "s3:GetBucketLocation",
```

```
            "s3:ListBucket"
        ],
        "Resource": "arn:aws:s3:::*rum*",
        "Condition": {
            "StringEquals": {
                "aws:ResourceAccount": "${aws:PrincipalAccount}"
            }
        }
    },
    {
        "Sid": "AllowS3BucketPolicyActions",
        "Effect": "Allow",
        "Action": [
            "s3:PutBucketPolicy",
            "s3:GetBucketPolicy"
        ],
        "Resource": "arn:aws:s3:::*rum*",
        "Condition": {
            "StringEquals": {
                "aws:ResourceAccount": "${aws:PrincipalAccount}"
            }
        }
    },
    {
        "Sid": "AllowS3ObjectActions",
        "Effect": "Allow",
        "Action": [
            "s3:GetObject",
            "s3:PutObject",
            "s3:DeleteObject",
            "s3:AbortMultipartUpload"
        ],
        "Resource": "arn:aws:s3:::*rum*",
        "Condition": {
            "StringEquals": {
                "aws:ResourceAccount": "${aws:PrincipalAccount}"
            }
        }
    }
```

- To use your own Amazon KMS keys for server-side encryption on your source map bucket, your IAM role or policy will need specific Amazon KMS permissions that allows creating a key, updating the key policy, using the Amazon KMS key with Amazon S3 and setting the encryption configuration of your Amazon S3 bucket. For security, scope these permissions to specific purposes. The example below restricts access to keys for a specific region and accountId and has similar S3 restrictions as the above example.

```
{
    "Sid": "AllowKMSKeyCreation",
    "Effect": "Allow",
    "Action": [
        "kms:CreateKey",
        "kms:CreateAlias"
    ],
    "Resource": "*"
},
{
    "Sid": "KMSReadPermissions",
    "Effect": "Allow",
    "Action": [
        "kms:ListAliases"
    ],
    "Resource": "*"
},
{
    "Sid": "AllowUpdatingKeyPolicy",
    "Effect": "Allow",
    "Action": [
        "kms:PutKeyPolicy",
        "kms:GetKeyPolicy",
        "kms:ListKeyPolicies"
    ],
    "Resource": "arn:aws:kms:REGION:ACCOUNT_ID:key/*"
},
{
    "Sid": "AllowUseOfKMSKeyForS3",
    "Effect": "Allow",
    "Action": [
        "kms:DescribeKey",
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:GenerateDataKey"
    ],
    "Resource": "arn:aws:kms:REGION:ACCOUNT_ID:key/*"
},
{
```

```
        "Sid": "AllowS3EncryptionConfiguration",
        "Effect": "Allow",
        "Action": [
            "s3:PutEncryptionConfiguration",
            "s3:GetEncryptionConfiguration"
        ],
        "Resource": "arn:aws:s3:::*rum*",
        "Condition": {
            "StringEquals": {
                "aws:ResourceAccount": "${aws:PrincipalAccount}"
            }
        }
    }
```

Other users who need to view CloudWatch RUM data but don't need to create CloudWatch RUM resources, can be granted the **AmazonCloudWatchRUMReadOnlyAccess** policy.

Set up an application to use CloudWatch RUM

Use the steps in these sections to set up your application to begin using CloudWatch RUM to collect performance data from real user sessions.

Topics

Step 1: Authorize your application to send data to Amazon

You have four options to set up data authentication:

- Use Amazon Cognito and let CloudWatch RUM create a new Amazon Cognito identity pool for the application. This method requires the least effort to set up.

  The identity pool will contain an unauthenticated identity. This allows the CloudWatch RUM web client to send data to CloudWatch RUM without authenticating the user of the application.

The Amazon Cognito identity pool has an attached IAM role. The Amazon Cognito unauthenticated identity allows the web client to assume the IAM role that is authorized to send data to CloudWatch RUM.

- Use Amazon Cognito for authentication. If you use this, you can use an existing Amazon Cognito identity pool or create a new one to use with this app monitor. If you use an existing identity pool, you must also modify the IAM role that is attached to the identity pool. Use this option for identity pools that support unauthenticated users. You can use identity pools only from the same Region.
- Use authentication from an existing identity provider that you have already set up. In this case, you must get credentials from the identity provider and your application must forward these credentials to the RUM web client.

  Use this option for identity pools that support only authenticated users.
- Use resource-based policies to manage access to your app monitor. This includes the ability to send unauthenticated requests to CloudWatch RUM without Amazon credentials. To learn more about resource based policies and RUM, see Using resource-based policies with CloudWatch RUM.

The following sections include more details about these options.

Use an existing Amazon Cognito identity pool

If you choose to use a Amazon Cognito identity pool, you specify the identity pool when you add the application to CloudWatch RUM. The pool must support enabling access to unauthenticated identities. You can use identity pools only from the same Region.

You also must add the following permissions to the IAM policy that is attached to the IAM role that is associated with this identity pool.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "rum:PutRumEvents",
            "Resource": "arn:aws:rum:[region]:[accountid]:appmonitor/[app monitor 
 name]"
        }
```

```
    ]
}
```

Amazon Cognito will then send the necessary security token to enable your application to access CloudWatch RUM.

Third-party provider

If you choose to use private authentication from a third-party provider, you must get credentials from the identity provider and forward them to Amazon. The best way to do this is by using a *security token vendor*. You can use any security token vendor, including Amazon Cognito with Amazon Security Token Service. For more information about Amazon STS, see Welcome to the Amazon Security Token Service API Reference.

If you want to use Amazon Cognito as the token vendor in this scenario, you can configure Amazon Cognito to work with an authentication provider. For more information, see Getting Started with Amazon Cognito Identity Pools (Federated Identities).

After you configure Amazon Cognito to work with your identity provider, you also need to do the following:

- Create an IAM role with the following permissions. Your application will use this role to access Amazon.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "rum:PutRumEvents",
            "Resource": "arn:aws:rum:[region]:[accountID]:appmonitor/[app monitor
 name]"
        }
    ]
}
```

- Add the following to your application to have it pass the credentials from your provider to CloudWatch RUM. Insert the line so that it runs after a user has signed in to your application and the application has received the credentials to use to access Amazon.

```
cwr('setAwsCredentials', {/* Credentials or CredentialProvider */});
```

For more information about credential providers in the Amazon JavaScript SDK, see Setting credentials in a web browser in the v3 developer guide for SDK for JavaScript, Setting credentials in a web browser in the v2 developer guide for SDK for JavaScript, , and @aws-sdk/credential-providers.

You can also use the SDK for the CloudWatch RUM web client to configure the web client authentication methods. For more information about the web client SDK, see CloudWatch RUM web client SDK.

Creating a CloudWatch RUM app monitor

To start using CloudWatch RUM with your application, you create an *app monitor*. When the app monitor is created, RUM generates a JavaScript snippet for you to paste into your application. The snippet pulls in the RUM web client code. The RUM web client captures data from a percentage of your application's user sessions and sends it to RUM.

To create an app monitor

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Application Signals**, **RUM**.
3. Choose **Add app monitor**.
4. Enter the information and settings for your application:

   - For **App monitor name**, enter a name to be used to identify this app monitor within the CloudWatch RUM console.
   - For **Application domain list**, enter the registered domain names where your application has administrative authority. You can also use a wildcard character * to allow any sub-domain or top-level domains (for example, *.amazon.com, amazon.*, *.amazon.*).
5. For **Configure RUM data collection**, specify whether you want the app monitor to collect each of the following:

   - **Performance telemetry** – Collects information about page load and resource load times
   - **JavaScript errors** – Collects information about unhandled JavaScript errors raised by your application

     You can select **Unminify JavaScript error stack traces** to debug unminified JavaScript errors. To use this feature, upload your source map files to an Amazon S3 bucket or folder and provide the Amazon S3 URI. Once enabled, RUM will use these source maps and enrich

JavaScript error events by adding the unminified stack trace. Note that after enabling, this feature only processes new JavaScript error events and cannot be used on previously collected data. For more information, see Enabling unminification of JavaScript error stack traces.

- **HTTP errors –** Collects information about HTTP errors thrown by your application

Selecting these options provides more information about your application, but also generates more CloudWatch RUM events and thus incurs more charges.

If you don't select any of these, the app monitor still collects session start events and page IDs so that you can see how many users are using your application, including breakdowns by operating system type and version, browser type and version, device type, and location.

6. Select **Check this option to allow the CloudWatch RUM Web Client to set cookies** if you want to be able to collect user IDs and session IDs from sampled user sessions. The user IDs are randomly generated by RUM. For more information, see CloudWatch RUM web client cookies (or similar technologies).
7. For **Session samples**, enter the percentage of user sessions that will be used to gather RUM data. The default is 100%. Reducing this number gives you less data, but reduces your charges. For more information about RUM pricing, see RUM pricing.
8. End user data that you collect for CloudWatch RUM is retained for 30 days and then deleted. If you want to keep copies of RUM events in CloudWatch Logs and configure how long to retain these copies, choose **Check this option to store your application telemetry data in your CloudWatch Logs account** under **Data storage**. By default, the CloudWatch Logs log group retains the data for 30 days. You can adjust the retention period in the CloudWatch Logs console.
9. (Optional) Choose to add a resource-based policy to your app monitor to control who can send `PutRumEvents` requests to your app monitor. If you choose **Create public policy**, a resource policy will be attached to your app monitor that enables anyone to send `PutRumEvents` requests to your app monitor. For more information about this method, see Using resource-based policies with CloudWatch RUM.
10. If you attached a resource-based policy in step 9, then you don't need to sign requests to CloudWatch RUM with Amazon credentials, and you can skip setting up authorization. Otherwise, for **Authorization**, specify whether to use a new or existing Amazon Cognito identity pool or use a different identity provider. Creating a new identity pool is the simplest

option that requires no other setup steps. For more information, see see Step 1: Authorize your application to send data to Amazon.

Creating a new Amazon Cognito identity pool requires administrative permissions. For more information, see IAM policies to use CloudWatch RUM.

11. (Optional) By default, when you add the RUM code snippet to your application, the web client injects the JavaScript tag to monitor usage into the HTML code of all pages of your application. To change this, choose **Configure pages** and then choose either **Include only these pages** or **Exclude these pages**. Then, specify the pages to include or exclude. To specify a page to include or exclude, enter its complete URLs. To specify additional pages, choose **Add URL**.
12. To enable Amazon X-Ray tracing of the user sessions that are sampled by the app monitor, choose **Active tracing** and select **Trace my service with Amazon X-Ray.**

    If you select this, `XMLHttpRequest` and `fetch` requests made during user sessions sampled by the app monitor are traced. You can then see traces and segments from these user sessions in the RUM dashboard, and the X-Ray trace map and trace details pages. These user sessions will also show up as client pages in Application Signals after you have enabled it for your application.

    By making additional configuration changes to the CloudWatch RUM web client, you can add an X-Ray trace header to HTTP requests to enable end-to-end tracing of user sessions through to downstream Amazon managed services. For more information, see Enabling X-Ray end-to-end tracing.
13. (Optional) To add tags to the app monitor, choose **Tags**, **Add new tag**.

    Then, for **Key**, enter a name for the tag. You can add an optional value for the tag in **Value**.

    To add another tag, choose **Add new tag** again.

    For more information, see Tagging Amazon Resources.
14. Choose **Add app monitor**.
15. In the **Sample code** section, you can copy the code snippet to use to add to your application. We recommend that you choose **JavaScript** or **TypeScript** and use NPM to install the CloudWatch RUM web client as a JavaScript module.

Alternatively, you can choose **HTML** to use a content delivery network (CDN) to install the CloudWatch RUM web client. The disadvantage of using a CDN is that the web client is often blocked by ad blockers.

16. Choose **Copy** or **Download**, and then choose **Done**.

Modifying the code snippet to configure the CloudWatch RUM web client (optional)

You can modify the code snippet before inserting it into your application, to activate or deactivate several options. For more information, see the [CloudWatch RUM web client documentation.]

There are four configuration options that you should definitely be aware of, as discussed in these sections.

Preventing the collection of resource URLs that might contain personal information

By default, the CloudWatch RUM web client is configured to record the URLs of resources downloaded by the application. These resources include HTML files, images, CSS files, JavaScript files, and so on. For some applications, URLs may contain personally identifiable information (PII).

If this is the case for your application, we strongly recommend that you disable the collection of resource URLs by setting `recordResourceUrl: false` in the code snippet configuration, before inserting it into your application.

Manually recording page views

By default, the web client records page views when the page first loads and when the browser's history API is called. The default page ID is `window.location.pathname`. However, in some cases you might want to override this behavior and instrument the application to record page views programmatically. Doing so gives you control over the page ID and when it is recorded. For example, consider a web application that has a URI with a variable identifier, such as `/entity/123` or `/entity/456`. By default, CloudWatch RUM generates a page view event for each URI with a distinct page ID matching the pathname, but you might want to group them by the same page ID instead. To accomplish this, disable the web client's page view automation by using the `disableAutoPageView` configuration, and use the `recordPageView` command to set the desired page ID. For more information, see [Application-specific Configurations] on GitHub.

Embedded script example:

```
cwr('recordPageView', { pageId: 'entityPageId' });
```

JavaScript module example:

```
awsRum.recordPageView({ pageId: 'entityPageId' });
```

Enabling X-Ray end-to-end tracing

When you create the app monitor, selecting **Trace my service with Amazon X-Ray** enables the tracing of `XMLHttpRequest` and `fetch` requests made during user sessions that are sampled by the app monitor. You can then see traces from these HTTP requests in the CloudWatch RUM dashboard, and the X-Ray Trace Map and Trace details pages.

By default, these client-side traces are not connected to downstream server-side traces. To connect client-side traces to server-side traces and enable end-to-end tracing, set the `addXRayTraceIdHeader` option to `true` in the web client. This causes the CloudWatch RUM web client to add an X-Ray trace header to HTTP requests.

The following code block shows an example of adding client-side traces. Some configuration options are omitted from this sample for readibility.

```
<script>
    (function(n,i,v,r,s,c,u,x,z){...})(
        'cwr',
        '00000000-0000-0000-0000-000000000000',
        '1.0.0',
        'us-west-2',
        'https://client.rum.us-east-1.amazonaws.com/1.0.2/cwr.js',
        {
            enableXRay: true,
            telemetries: [
                'errors',
                'performance',
                [ 'http', { addXRayTraceIdHeader: true } ]
            ]
        }
    );
</script>
```

> **Warning**
>
> Configuring the CloudWatch RUM web client to add an X-Ray trace header to HTTP requests can cause cross-origin resource sharing (CORS) to fail or invalidate the request's signature if the request is signed with SigV4. For more information, see the CloudWatch RUM web client documentation. We strongly recommend that you test your application before adding a client-side X-Ray trace header in a production environment.

For more information, see the CloudWatch RUM web client documentation

Sending unsigned requests to CloudWatch RUM

By default, the RUM web client signs all requests sent to RUM. If you set `signing:false` in the client configuration, requests will be unsigned when they are sent to CloudWatch RUM. Data will be ingested to RUM only if there is a public resource based policy attached to the app monitor. For more information, see Using resource-based policies with CloudWatch RUM.

Inserting the CloudWatch app monitor code snippet into your application

Next, you insert the code snippet that you created in the previous section into your application.

> **Warning**
>
> The web client, downloaded and configured by the code snippet, uses cookies (or similar technologies) to help you collect end user data. Before you insert the code snippet, see Filtering by metadata attributes in the console.

If you don't have the code snippet that was previously generated, you can find it by following the directions in How do I find a code snippet that I've already generated?.

To insert the CloudWatch RUM code snippet into your application

1. Insert the code snippet that you copied or downloaded in the previous section inside the `<head>` element of your application. Insert it before the `<body>` element or any other `<script>` tags.

   The following is an example of a generated code snippet:

```
<script>
(function (n, i, v, r, s, c, x, z) {
    x = window.AwsRumClient = {q: [], n: n, i: i, v: v, r: r, c: c};
    window[n] = function (c, p) {
        x.q.push({c: c, p: p});
    };
    z = document.createElement('script');
    z.async = true;
    z.src = s;
    document.head.insertBefore(z, document.getElementsByTagName('script')[0]);
})('cwr',
    '194a1c89-87d8-41a3-9d1b-5c5cd3dafbd0',
    '1.0.0',
    'us-east-2',
    'https://client.rum.us-east-1.amazonaws.com/1.0.2/cwr.js',
    {
        sessionSampleRate: 1,
        identityPoolId: "us-east-2:c90ef0ac-e3b8-4d1a-b313-7e73cfd21443",
        endpoint: "https://dataplane.rum.us-east-2.amazonaws.com",
        telemetries: ["performance", "errors", "http"],
        allowCookies: true,
        enableXRay: false
    });
</script>
```

2. If your application is a multipage web application, you must repeat step 1 for each HTML page that you want included in the data collection.

Testing your CloudWatch app monitor setup by generating user events

After you have inserted the code snippet and your updated application is running, you can test it by manually generating user events. To test this, we recommend that you do the following. This testing incurs standard CloudWatch RUM charges.

- Navigate between pages in your web application.
- Create multiple user sessions, using different browsers and devices.
- Make requests.
- Cause JavaScript errors.

After you have generated some events, view them in the CloudWatch RUM dashboard. For more information, see Viewing the CloudWatch RUM dashboard.

Data from user sessions might take up to 15 minutes to appear in the dashboard.

If you don't see data 15 minutes after you generated events in the application, see Troubleshooting CloudWatch RUM.

Using resource-based policies with CloudWatch RUM

You can attach a resource policy to a CloudWatch RUM app monitor. By default, app monitors do not have a resource policy attached to them. CloudWatch RUM resource based policies do not support cross-account access.

To learn more about Amazon resource policies, see Identity-based policies and resource-based policies.

To learn more about how resource policies and identity policies are evaluated, see Policy evaluation logic.

To learn more about IAM policy grammar, see IAM JSON policy element reference.

Supported actions

Resource-based policies on app monitors support the `rum:PutRumEvents` action.

Sample policies to use with CloudWatch RUM

The following example allows anyone to write data to your app monitor, including those without SigV4 credentials.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "rum:PutRumEvents",
            "Resource": "arn:aws:rum:region:accountID:appmonitor/app monitor name",
            "Principal" : "*"

        }
```

```
    ]
}
```

You can modify the policy to block specified source IP addresses by using the `aws:SourceIp` condition key. With this example, Using this policy, PutRumEvents from the IP address listed will be rejected. All other requests from other IP addresses will be accepted. For more information about this condition key, see Properties of the network in the IAM User Guide.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "rum:PutRumEvents",
            "Resource": "arn:aws:rum:region:accountID:appmonitor/app monitor name",
            "Principal" : "*"

        },
        {
            "Effect": "Deny",
            "Action": "rum:PutRumEvents",
            "Resource": "arn:aws:rum:region:accountID:appmonitor/app monitor name",
            "Principal" : "*",
           "Condition": {
                "NotIpAddress": {
                    "aws:SourceIp": "************"
                }
            }
        }
    ]
}
```

Additionally, you can also choose to only accept PutRumEvents requests that are signed with a certain alias using the `rum:alias` service context key. In the following example, `PutRumEvents` will have to set the optional `Alias` request parameter to either `alias1` or `alias2` for the event to be accepted. To configure your web client to send `Alias` you must use version 1.20 or later of the CloudWatch RUM web client, as described in Application-specific Configurations on GitHub.

```
{
    "Version": "2012-10-17",
    "Statement": [
```

```
        {
            "Effect": "Allow",
            "Action": "rum:PutRumEvents",
            "Resource": "arn:aws:rum:region:accountID:appmonitor/app monitor name",
            "Principal" : "*",
             "Condition": {
             "StringEquals": { "rum:alias": [ "alias1", "alias2"] } }
        }
    ]
}
```

Configuring the CloudWatch RUM web client

Your applications can use one of the code snippets generated by CloudWatch RUM to install the CloudWatch RUM web client. The generated snippets support two installation methods: as a JavaScript module through NPM, or from a content delivery network (CDN). For best performance, we recommend using the NPM installation method. For more information about using this method, see [Installing as a JavaScript Module](#).

If you use the CDN installation option, ad blockers might block the default CDN provided by CloudWatch RUM. This disables application monitoring for users who have ad blockers installed. Because of this, we recommend that you use the default CDN only for initial onboarding with CloudWatch RUM. For more information about the ways to mitigate this issue, see [Instrument the application](#).

The code snippet sits in the `<head>` tag of an HTML file and installs the web client by downloading the web client, and then configuring the web client for the application it is monitoring. The snippet is a self-executing function which looks similar to the following. In this example, the body of the snippet's function has been omitted for readability.

```
<script>
(function(n,i,v,r,s,c,u,x,z){...})(
'cwr',
'00000000-0000-0000-0000-000000000000',
'1.0.0',
'us-west-2',
'https://client.rum.us-east-1.amazonaws.com/1.0.2/cwr.js',
{ /* Configuration Options Here */ }
);
<script>
```

Arguments

The code snippet accepts six arguments:

- A namespace for running commands on the web client, such as `'cwr'`
- The ID of the app monitor, such as `'00000000-0000-0000-0000-000000000000'`
- The application version, such as `'1.0.0'`
- The Amazon Region of the app monitor, such as `'us-west-2'`
- The URL of the web client, such as `'https://client.rum.us-east-1.amazonaws.com/1.0.2/cwr.js'`
- Application-specific configuration options. For more information, see the following section.

Ignoring errors

The CloudWatch RUM web client listens to all types of errors that happen in your applications. If your application emits JavaScript errors that you do not want to view in the CloudWatch RUM dashboard, you can configure the CloudWatch RUM web client to filter out these errors so that you see only the relevant error events on the CloudWatch RUM dashboard. For example, you might choose not to view some JavaScript errors in the dashboard because you have already identified a fix for them and the volume of these errors is masking other errors. You might also want to ignore errors that you can't fix because they are owned by a library owned by a third party.

For more information about how to instrument the web client to filter out specific JavaScript errors, see the example in Errors in the web client Github documentation.

Configuration options

For information about the configuration options available for the CloudWatch RUM web client, see the CloudWatch RUM web client documentation

Enabling unminification of JavaScript error stack traces

When your web application JavaScript source code is minified, error stack traces can be difficult to read. You can enable unminification to the stack traces by uploading your source maps to Amazon S3. CloudWatch RUM will retrieve the source maps to map the line and column numbers in the minified source code back to the original unminified source code. This will improve readability of your error stack traces and help identify the location of the error in the original source code.

Requirements and syntax

Source maps are crucial for debugging and tracking issues in your web application across different releases. Make sure that each web application release has a unique source map. Each release should have its own unique releaseId. A releaseId must be a string between 1 and 200 characters long and can only contain letters, numbers, underscores, hyphens, colons, forward slashes, and periods. To add the `releaseId` as metadata to RUM events, configure the CloudWatch RUM web client.

Source maps are expected to be plain JSON files following the structure defined by the Source Map V3 specification. The required fields are: `version`, `file`, `sources`, `names`, and `mappings`.

Make sure the size of each source map does not exceed the limit of 50 MB. In addition, RUM service will only retrieve up to 50 MB of source maps per stack trace. If needed, split the source code into multiple smaller chunks. For more information, see Code Splitting with WebpackJS.

Topics

Configure your Amazon S3 bucket resource policy to allow RUM service access

Make sure your Amazon S3 bucket is in the same region as your RUM appMonitor. Configure your Amazon S3 bucket to allow RUM service access for retrieving source map files. Include the `aws:SourceArn` and `aws:SourceAccount` global condition context keys to limit the service's permissions to the resource. This is the most effective way to protect against the confused deputy problem.

The following example shows how you can use the `aws:SourceArn` and `aws:SourceAccount` global condition context keys in Amazon S3 to prevent the confused deputy problem.

```
{
    "Version": "2012-10-17",
    "Statement": [
```

```
        {
            "Sid": "RUM Service S3 Read Permissions",
            "Effect": "Allow",
            "Principal": {
                "Service": "rum.amazonaws.com"
            },
            "Action": [
                "s3:GetObject",
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::BUCKET_NAME",
                "arn:aws:s3:::BUCKET_NAME/*"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "ACCOUNT_ID",
                    "aws:SourceArn":
 "arn:aws:rum:REGION:ACCOUNT_ID:appmonitor/APP_MONITOR_NAME"
                }
            }
        }
    ]
}
```

If you are using Amazon KMS keys to encrypt the data, make sure the key's resource policy is configured similarly to include the `aws:SourceArn` and `aws:SourceAccount` global condition context keys to give RUM service access to use the keys to retrieve the source map files.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "RUM Service KMS Read Permissions",
            "Effect": "Allow",
            "Principal": {
                "Service": "rum.amazonaws.com"
            },
            "Action": "kms:Decrypt",
            "Resource": "arn:aws:kms:REGION:ACCOUNT_ID:key/KEY_ID",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "ACCOUNT_ID",
```

```
                    "aws:SourceArn": "arn:aws:rum:REGION:ACCOUNT_ID/APP_MONITOR_NAME"
                }
            }
        }
    ]
}
```

Upload source maps

Configure your JavaScript bundle to generate source maps during minification. When you build your application, the bundle will create a directory (for example, dist) containing the minified JavaScript files and their corresponding source maps. See below for an example.

```
./dist
    |-index.d5a07c87.js
    |-index.d5a07c87.js.map
```

Upload the source map files to your Amazon S3 bucket. The files should be located in a folder with the `releaseId` as the name. For example, if my bucket name is `my-application-source-maps` and the `releaseId` is 2.0.0, then the source map file is located at the following location:

```
my-application-source-maps
    |-2.0.0
        |-index.d5a07c87.js.map
```

To automate uploading your source maps, you can create the following bash script and execute it as part of your build process.

```
#!/bin/bash
# Ensure the script is called with required arguments
if [ "$#" -ne 2 ]; then
 echo "Usage: $0 S3_BUCKET_NAME RELEASE_ID"
 exit 1
fi

# Read arguments
S3_BUCKET="$1"
RELEASE_ID="$2"

# Set the path to your build directory
BUILD_DIR="./dist"
```

```
# Upload all .map files recursively
 if aws s3 cp "$BUILD_DIR" "s3://$S3_BUCKET/$RELEASE_ID/" --recursive --exclude "*" --
include "*.map"; then
    echo "Successfully uploaded all source map files"
else
    echo "Failed to upload source map files"
fi
```

Configure releaseId in your CloudWatch RUM web client

CloudWatch RUM uses the configured `releaseId` to determine the folder to retrieve the source map files. Name the `releaseId` the same as your source map files folder. If you used the provided bash script above or a similar one, the `releaseId` configured in the script should be the same as the one configured in your CloudWatch RUM web client. You must use version 1.21.0 or later of the CloudWatch RUM web client.

```
import { AwsRum, AwsRumConfig } from "aws-rum-web";

try {
    const config: AwsRumConfig = {
        sessionSampleRate: 1,
        endpoint: "https://dataplane.rum.us-west-2.amazonaws.com",
        telemetries: ["performance", "errors", "http"],
        allowCookies: true,
        releaseId: "RELEASE_ID", //Add this
    };

    const APPLICATION_ID: string = "APP_MONITOR_ID";
    const APPLICATION_VERSION: string = "1.0.0";
    const APPLICATION_REGION: string = "us-west-2";

    new AwsRum(APPLICATION_ID, APPLICATION_VERSION, APPLICATION_REGION, config);
} catch (error: any) {
    // Ignore errors thrown during CloudWatch RUM web client initialization
}
```

Enabling CloudWatch RUM app monitor to unminify JavaScript stack traces

To unminify JavaScript stack traces, set the app monitor's SourceMap status to ENABLED. Provide the Amazon S3 URI to the bucket or folder containing all source maps for your app monitor.

When storing source maps directly in the main bucket (not in a subfolder), then the Amazon S3 URI should be formatted as `Amazon S3://BUCKET_NAME`. In this case, source map files should be located at the following location.

```
BUCKET_NAME
    |- RELEASE_ID
        |-index.d5a07c87.js.map
```

When a child directory is the root, then the Amazon S3 URI should be formatted as `Amazon S3://BUCKET_NAME/DIRECTORY`. In this case, source map files should be located at the following location.

```
BUCKET_NAME
    |- DIRECTORY
        |-RELEASE_ID
            |-index.d5a07c87.js.map
```

Viewing unminified stack traces in the RUM console

After uploading your source maps to Amazon S3, enabling source maps on your RUM app monitor, and deploying your web application with the `releaseId` configured in the CloudWatch RUM web client, select **Events** in the RUM console. This tab displays the raw RUM event data. Filter by the JS error event type and view the latest JS error event. You will see the unminified stack trace in the new `event_details.unminifiedStack` field for events ingested after the feature was enabled.

Viewing unminified stack traces in CloudWatch Logs

Enable RUM event storage in CloudWatch Logs by turning on **Data storage**. Once enabled, you can search the new **event_details.unminifiedStack** field. This allows you to analyze trends and relate issues across multiple sessions using CloudWatch Logs queries.

Troubleshooting source maps

CloudWatch RUM provides out of the box metrics to troubleshoot your source map setup. These metrics are published in the metric namespace named `AWS/RUM`. The following metrics are published with an application_name dimension. The value of this dimension is the name of the app monitor. The metrics are also published with an `aws:releaseId` dimension. The value of this dimension is the `releaseId` associated with the JavaScript error event.

MetricName	Unit	Description
UnminifyLineFailureCount	Count	The count of stack trace lines in the JS error event that failed to be unminified. Additional details regarding the failure will be added to the specific line that failed in the event_details.unmi nifiedStack field.
UnminifyLineSuccessCount	Count	The count of stack trace lines in the JS error event that were successfully unminified.
UnminifyEventFailureCount	Count	The count of JS error events that failed to have any lines unminified. Additional details regarding the failure will be added in the event_det ails.unminifiedStack field.
UnminifyEventSuccessCount	Count	The count of JS error events that succeeded to have at least one stack trace line unminified.

CloudWatch RUM may fail to unminify a line in the stack trace for various reasons, including but not limited to:

- Failure to retrieve corresponding source map file due to permission issues. Make sure the bucket resource policy is configured correctly.
- Corresponding source map file does not exist. Make sure the source map files have been uploaded to the correct bucket or folder that has the same name as the releaseId configured in your CloudWatch RUM web client.
- Corresponding source map file is too big. Split your source code into smaller chunks.

- 50 MB worth of source map files already retrieved for the stack trace. Reduce the stack trace length as 50 MB is service side limitation.
- Source map is invalid and could not be indexed. Make sure the source map is a plain JSON following the structure defined by the Source Map V3 specification and includes the following fields: version, file, sources, names, mappings.
- Source map could not map the minified source code back to the unminified stack trace. Make sure the source map is the correct source map for the given releaseId.

Regionalization

This section illustrates strategies for using CloudWatch RUM with applications in different Regions.

My web application is deployed in multiple Amazon Regions

If your web application is deployed in multipled Amazon Regions, you have three options:

- Deploy one app monitor in one Region, in one account, serving all Regions.
- Deploy separate app monitors for each Region, in unique accounts.
- Deploy separate app monitors for each Region, all in one account.

The advantage of using one app monitor is that all data will be centralized into one visualization, and all logs are written to the same log group in CloudWatch Logs. With a single app monitor there is a small amount of extra latency for requests, and a single point of failure.

Using multiple app monitors removes the single point of failure, but prevents all data from being combined into one visualization.

CloudWatch RUM hasn't launched in some Regions that my application is deployed in

CloudWatch RUM is launched into many Regions and has wide geographical coverage. By setting up CloudWatch RUM in the Regions where it is available, you can get the benefits. End users can be anywhere and still have their sessions included if you have set up an app monitor in the Region that they are connecting to.

However, CloudWatch RUM is not yet launched in Amazon GovCloud (US-East), Amazon GovCloud (US-West), or any Regions in China. You are not able to send data to CloudWatch RUM from these Regions.

Use page groups

Use page groups to associate different pages in your application with each other so that you can see aggregated analytics for groups of pages. For example, you might want to see the aggregated page load times of all of your landing pages.

You put pages into page groups by adding one or more tags to page view events in the CloudWatch RUM web client. The following examples put the `/home` page into the page group named `en` and the page group named `landing`.

Embedded script example

```
cwr('recordPageView', { pageId: '/home', pageTags: ['en', 'landing']});
```

JavaScript module example

```
awsRum.recordPageView({ pageId: '/home', pageTags: ['en', 'landing']});
```

> **Note**
>
> Page groups are intended to facilitate aggregating analytics across different pages. For information about how to define and manipulate `pageIds` for your application, see the **Manually recording page views** section in Modifying the code snippet to configure the CloudWatch RUM web client (optional).

Specify custom metadata

CloudWatch RUM attaches additional data to each event as metadata. Event metadata consists of attributes in the form of key-value pairs. You can use these attributes to search or filter events in the CloudWatch RUM console. By default, CloudWatch RUM creates some metadata for you. For more information about the default metadata, see RUM event metadata.

You can also use the CloudWatch RUM web client to add custom metadata to CloudWatch RUM events. The custom metadata can include session attributes and page attributes.

To add custom metadata, you must use version 1.10.0 or later of the CloudWatch RUM web client.

Requirements and syntax

Each event can include as many as 10 custom attributes in the metadata. The syntax requirements for custom attributes are as follows:

- **Keys**
  - Maximum of 128 characters
  - Can include alphanumeric characters, colons (:), and underscores (_)
  - Can't begin with `aws:`.
  - Can't consist entirely of any of the reserved keywords listed in the following section. Can use those keywords as part of a longer key name.
- **Values**
  - Maximum of 256 characters
  - Must be strings, numbers, or Boolean values

Reserved keywords

You can't use the following reserved keywords as complete key names. You can use the following keywords as part of a longer key name, such as `applicationVersion`.

- `browserLanguage`
- `browserName`
- `browserVersion`
- `countryCode`
- `deviceType`
- `domain`
- `interaction`
- `osName`
- `osVersion`
- `pageId`
- `pageTags`
- `pageTitle`
- `pageUrl`

- parentPageId
- platformType
- referrerUrl
- subdivisionCode
- title
- url
- version

> **Note**
>
> CloudWatch RUM removes custom attributes from RUM events if an attribute includes a key or value that is not valid, or if the limit of 10 custom attributes per event has already been reached.

Add session attributes

If you configure custom session attributes, they are added to all events in a session. You configure session attributes either during CloudWatch RUM web client initialization or at runtime by using the addSessionAttributes command.

For example, you can add your application's version as a session attribute. Then, in the CloudWatch RUM console, you can filter errors by version to find whether an increased error rate is associated with a particular version of your application.

Adding a session attribute at initialization, NPM example

The code section in bold adds the session attribute.

```
import { AwsRum, AwsRumConfig } from 'aws-rum-web';

try {
  const config: AwsRumConfig = {
    allowCookies: true,
    endpoint: "https://dataplane.rum.us-west-2.amazonaws.com",
    guestRoleArn: "arn:aws:iam::000000000000:role/RUM-Monitor-us-
west-2-000000000000-00xx-Unauth",
```

```
    identityPoolId: "us-west-2:00000000-0000-0000-0000-000000000000",
    sessionSampleRate: 1,
    telemetries: ['errors', 'performance'],
    sessionAttributes: {
        applicationVersion: "1.3.8"
    }
  };

  const APPLICATION_ID: string = '00000000-0000-0000-0000-000000000000';
  const APPLICATION_VERSION: string = '1.0.0';
  const APPLICATION_REGION: string = 'us-west-2';

  const awsRum: AwsRum = new AwsRum(
    APPLICATION_ID,
    APPLICATION_VERSION,
    APPLICATION_REGION,
    config
  );
} catch (error) {
  // Ignore errors thrown during CloudWatch RUM web client initialization
}
```

Adding a session attribute at runtime, NPM example

```
awsRum.addSessionAttributes({
    applicationVersion: "1.3.8"
})
```

Adding a session attribute at initialization, embedded script example

The code section in bold adds the session attribute.

```
<script>
    (function(n,i,v,r,s,c,u,x,z){...})(
        'cwr',
        '00000000-0000-0000-0000-000000000000',
        '1.0.0',
        'us-west-2',
        'https://client.rum.us-east-1.amazonaws.com/1.0.2/cwr.js',
        {
            sessionSampleRate:1,
            guestRoleArn:'arn:aws:iam::000000000000:role/RUM-Monitor-us-
west-2-000000000000-00xx-Unauth',
```

```
            identityPoolId:'us-west-2:00000000-0000-0000-0000-000000000000',
            endpoint:'https://dataplane.rum.us-west-2.amazonaws.com',
            telemetries:['errors','http','performance'],
            allowCookies:true,
            sessionAttributes: {
                applicationVersion: "1.3.8"
            }
        }
    );
</script>
```

Adding a session attribute at runtime, embedded script example

```
<script>
    function addSessionAttribute() {
        cwr('addSessionAttributes', {
            applicationVersion: "1.3.8"
        })
    }

</script>
```

Add page attributes

If you configure custom page attributes, they are added to all events on the current page. You configure page attributes either during CloudWatch RUM web client initialization or at runtime by using the `recordPageView` command.

For example, you can add your page template as a page attribute. Then, in the CloudWatch RUM console, you can filter errors by page templates to find whether an increased error rate is associated with a particular page template of your application.

Adding a page attribute at initialization, NPM example

The code section in bold adds the page attribute.

```
const awsRum: AwsRum = new AwsRum(
    APPLICATION_ID,
    APPLICATION_VERSION,
    APPLICATION_REGION,
    { disableAutoPageView:  true // optional }
);
```

```
awsRum.recordPageView({
    pageId:'/home',
    pageAttributes: {
      template: 'artStudio'
    }
});
const credentialProvider = new CustomCredentialProvider();
if(awsCreds) awsRum.setAwsCredentials(credentialProvider);
```

Adding a page attribute at runtime, NPM example

```
awsRum.recordPageView({
    pageId: '/home',
    pageAttributes: {
        template: 'artStudio'
    }
});
```

Adding a page attribute at initialization, embedded script example

The code section in bold adds the page attribute.

```
<script>
    (function(n,i,v,r,s,c,u,x,z){...})(
        'cwr',
        '00000000-0000-0000-0000-000000000000',
        '1.0.0',
        'us-west-2',
        'https://client.rum.us-east-1.amazonaws.com/1.0.2/cwr.js',
        {
            disableAutoPageView: true //optional
        }
    );
    cwr('recordPageView', {
       pageId: '/home',
       pageAttributes: {
           template: 'artStudio'
       }
    });
    const awsCreds = localStorage.getItem('customAwsCreds');
    if(awsCreds) cwr('setAwsCredentials', awsCreds)
</script>
```

Adding a page attribute at runtime, embedded script example

```
<script>
    function recordPageView() {
        cwr('recordPageView', {
            pageId: '/home',
            pageAttributes: {
                template: 'artStudio'
            }
        });
    }
</script>
```

Filtering by metadata attributes in the console

To filter the visualizations in the CloudWatch RUM console with any built-in or custom metadata attribute, use the search bar. In the search bar, you can specify as many as 20 filter terms in the form of **key=value** to apply to the visualizations. For example, to filter data for only the Chrome browser, you could add the filter term **browserName=Chrome**.

By default, the CloudWatch RUM console retrieves the 100 most common attributes keys and values to display in the dropdown in the search bar. To add more metadata attributes as filter terms, enter the complete attribute key and value into the search bar.

A filter can include as many as 20 filter terms, and you can save up to 20 filters per app monitor. When you save a filter, it is saved in the **Saved filters** dropdown. You can also delete a saved filter.

Send custom events

CloudWatch RUM records and ingests the events listed in Information collected by the CloudWatch RUM web client. If you use version 1.12.0 or later of the CloudWatch RUM web client, you can define, record, and send additional custom events. You define the event type name and the data to send for each event type that you define. Each custom event payload can be up to 6 KB.

Custom events are ingested only if the app monitor has custom events enabled. To update the configuration settings of your app monitor, use the CloudWatch RUM console or the UpdateAppMonitor API.

After you enable custom events, and then define and send custom events, you can search for them. To search for them, use the **Events** tab in the CloudWatch RUM console. Search by using the event type.

Requirements and syntax

Custom events consist of an event type and event details. The requirements for these are as follows:

- **Event type**
  - This can be either the **type** or **name** of your event. For example, the CloudWatch RUM built-in event type called **JsError** has an event type of `com.amazon.rum.js_error_event`.
  - Must be between 1 and 256 characters.
  - Can be a combination of alphanumeric characters, underscores, hyphens, and periods.
- **Event details**
  - Contains the actual data that you want to record in CloudWatch RUM.
  - Must be an object that consists of fields and values.

Examples of recording custom events

There are two ways to record custom events in the CloudWatch RUM web client.

- Use the CloudWatch RUM web client's `recordEvent` API.
- Use a customized plugin.

Send a custom event using the `recordEvent` API, NPM example

```
awsRum.recordEvent('my_custom_event', {
        location: 'IAD',
        current_url: 'amazonaws.com',
        user_interaction: {
            interaction_1 : "click",
            interaction_2 : "scroll"
        },
        visit_count:10
    }
)
```

Send a custom event using the `recordEvent` API, embedded script example

```
cwr('recordEvent', {
    type: 'my_custom_event',
```

```
    data: {
        location: 'IAD',
        current_url: 'amazonaws.com',
        user_interaction: {
            interaction_1 : "click",
            interaction_2 : "scroll"
        },
        visit_count:10
    }
})
```

Example of sending a custom event using a customized plugin

```
// Example of a plugin that listens to a scroll event, and
// records a 'custom_scroll_event' that contains the timestamp of the event.
class MyCustomPlugin implements Plugin {
    // Initialize MyCustomPlugin.
    constructor() {
        this.enabled;
        this.context;
        this.id = 'custom_event_plugin';
    }
    // Load MyCustomPlugin.
    load(context) {
        this.context = context;
        this.enable();
    }
    // Turn on MyCustomPlugin.
    enable() {
        this.enabled = true;
        this.addEventHandler();
    }
    // Turn off MyCustomPlugin.
    disable() {
        this.enabled = false;
        this.removeEventHandler();
    }
    // Return MyCustomPlugin Id.
    getPluginId() {
        return this.id;
    }
    // Record custom event.
    record(data) {
```

```
        this.context.record('custom_scroll_event', data);
    }
    // EventHandler.
    private eventHandler = (scrollEvent: Event) => {
        this.record({timestamp: Date.now()})
    }
    // Attach an eventHandler to scroll event.
    private addEventHandler(): void {
        window.addEventListener('scroll', this.eventHandler);
    }
    // Detach eventHandler from scroll event.
    private removeEventHandler(): void {
        window.removeEventListender('scroll', this.eventHandler);
    }
}
```

Viewing the CloudWatch RUM dashboard

CloudWatch RUM helps you collect data from user sessions about your application's performance, including page load times, Apdex score, browsers and devices used, geolocation of user sessions, and sessions with errors. All of this information is displayed in a dashboard.

To view the RUM dashboard

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Application Signals**, **RUM**.

   The **Overview** tab displays information collected by one of the app monitors that you have created.

   The top row of panes displays the following information for this app monitor:

   - Number of page loads
   - Average page load speed
   - Apdex score
   - Status of any alarms associated with the app monitor

   The application performance index (Apdex) score indicates end users' level of satisfaction. Scores range from 0 (least satisfied) to 1 (most satisfied). The scores are based on application performance only. Users are not asked to rate the application. For more information about Apdex scores, see How CloudWatch RUM sets Apdex scores.

Several of these panes include links that you can use to further examine the data. Choosing any of these links displays a detailed view with **Performance**, **Errors**, **HTTP requests**, **Sessions**, **Events Browsers & Devices**, and **User Journey** tabs at the top of the display.

3. To focus further, choose the **List view** tab and then choose the name of the app monitor that you want to focus on. This displays the following tabs for the chosen app monitor.

   - The **Performance** tab displays page performance information including load times, request information, web vitals, and page loads over time. This view features interactive web vitals graphs where you can see the different percentile values of core web vitals for your pages and choose datapoints on the graph to view associated events captured by CloudWatch RUM. From there, you can either explore more events related to the metric spike or view page details for a selected event to identify specific conditions causing performance issues.

     On this tab you can also toggle the view between **Page loads**, **Requests**, and **Location** to see more details about page performance.
   - The **Errors** tab displays Javascript error information including the error message most frequently seen by users and the devices and browsers with the most errors. This view includes a histogram of the errors and a list view of errors. You can filter the list of errors by user and event details. Choose an error message to see more details.
   - The **HTTP requests** tab displays HTTP request information including the request URL with most errors and the devices and browsers with the most errors. This tab includes a histogram of the requests, a list view of requests, and a list view of network errors. You can filter the lists by user and event details. Choose a response code or an error message to s ee more details about the request or network error, respectively.
   - The **Sessions** tab displays session metrics. This tab includes a histogram of session start events and a list view of sessions. You can filter the list of sessions by event type, user details, and event details. Choose a **sessionId** to see more details about a session.
   - The **Events** tab displays a histogram of RUM events and a list view of the events. You can filter the list of events by event type, user details, and event details. Choose a RUM event to see the raw event.
   - The **Browsers & Devices** tab displays information such as the performance and usage of different browsers and devices to access your application. This view includes controls to toggle the view between focusing on **Browsers** and **Devices**.

If you narrow the scope to a single browser, you see the data broken down by browser version.

- The **User Journey** tab displays the paths that your customers use to navigate your application. You can see where your customers enter your application and what page they exit your application from. You can also see the paths that they take and the percentage of customers that follow those paths. You can pause on a node to get more details about that page. You can choose a single path to highlight the connections for easier viewing.

4. (Optional) On any of the first six tabs, you can choose the **Pages** button and select a page or page group from the list. This narrows down the displayed data to a single page or group of pages of your application. You can also mark pages and page groups in the list as favorites.

How CloudWatch RUM sets Apdex scores

Apdex (Application Performance Index) is an open standard that defines a method to report, benchmark, and rate application response time. An Apdex score helps you understand and identify the impact on application performance over time.

The Apdex score indicates the end users' level of satisfaction Scores range from 0 (least satisfied) to 1 (most satisfied). The scores are based on application performance only. Users are not asked to rate the application.

Each individual Apdex score falls into one of three thresholds. Based on the Apdex threshold and actual application response time, there are three kinds of performance, as follows:

- **Satisfied**– The actual application response time is less than or equal to the Apdex threshold. For CloudWatch RUM, this threshold is 2000 ms or less.
- **Tolerable**– The actual application response time is greater than the Apdex threshold, but less than or equal to four times the Apdex threshold. For CloudWatch RUM, this range is 2000–8000 ms.
- **Frustrating**– The actual application response time is greater than four times the Apdex threshold. For CloudWatch RUM, this range is over 8000 ms.

The total 0-1 Apdex score is calculated using the following formula:

```
(positive scores + tolerable scores/2)/total scores * 100
```

CloudWatch metrics that you can collect with CloudWatch RUM

The table in this section lists the metrics that you automatically collect with CloudWatch RUM. You can see these metrics in the CloudWatch console. For more information, see View available metrics.

You can also optionally send extended metrics to CloudWatch or CloudWatch Evidently. For more information, see Extended metrics.

These metrics are published in the metric namespace named `AWS/RUM`. All of the following metrics are published with an `application_name` dimension. The value of this dimension is the name of the app monitor. Some metrics are also published with additional dimensions, as listed in the table.

Metric	Unit	Description
`HttpStatusCodeCount`	Count	The count of HTTP responses in the application, by their response status code. Additional dimension s: • `event_details.response.status` is the response status code, such as 200, 400, 404, and so on. • `event_type` The type of event. Currently, the only possible value for this dimension is `http`.
`Http4xxCount`	Count	The count of HTTP responses in the

Metric	Unit	Description
		application, with 4xx response status code. These are calculated based on `http_event` RUM events that result in 4xx codes.
`Http4xxCountPerSession`	Count	The count of HTTP responses in a session, with 4xx response status code. These are calculated based on `http_event` RUM events that result in 4xx codes.
`Http4xxCountPerPageView`	Count	The count of HTTP responses in a page review, with 4xx response status code. These are calculated based on `http_event` RUM events that result in 4xx codes.
`Http5xxCount`	Count	The count of HTTP responses in the application, with 5xx response status code. These are calculated based on `http_event` RUM events that result in 5xx codes.

Metric	Unit	Description
Http5xxCountPerSession	Count	The count of HTTP responses in the session, with 5xx response status code. These are calculated based on http_event RUM events that result in 5xx codes.
Http5xxCountPerPageView	Count	The count of HTTP responses in a page review, with 5xx response status code. These are calculated based on http_event RUM events that result in 5xx codes.
JsErrorCount	Count	The count of JavaScript error events ingested.
JsErrorCountPerSession	Count	The count of JavaScript error events ingested in a session.
JsErrorCountPerPageView	Count	The count of JavaScript error events ingested in a page review.

Metric	Unit	Description
`NavigationFrustratedTransaction`	Count	The count of navigation events with a `duration` higher than the frustrating threshold, which is 8000ms. The duration of navigation events is tracked in the `PerformanceNavigationDuration` metric.
`NavigationSatisfiedTransaction`	Count	The count of navigation events with a `duration` that is less than the Apdex objective, which is 2000ms. The duration of navigation events is tracked in the `PerformanceNavigationDuration` metric.

Metric	Unit	Description
`NavigationToleratedTransaction`	Count	The count of navigation events with a `duration` between 2000ms and 8000ms. The duration of navigation events is tracked in the `PerformanceNavigationDuration` metric.
`PageViewCount`	Count	The count of page view events ingested by the app monitor. This is calculated by counting the `page_view_event` RUM events.
`PageViewCountPerSession`	Count	The count of page view events in a session.

Metric	Unit	Description
`PerformanceResourceDuration`	Milliseconds	The `duration` of a resource event. Additional dimensions: • `event_details.file.type` is the file type of the resource event, such as a stylesheet, document, image, script, or font. • `event_type` The type of event. Currently, the only possible value for this dimension is `resource`.
`PerformanceNavigationDuration`	Milliseconds	The `duration` of a navigation event.
`RumEventPayloadSize`	Bytes	The size of every event ingested by CloudWatch RUM. You can also use the `SampleCount` statistic for this metric to monitor the number of events that an app monitor is ingesting.

Metric	Unit	Description
SessionCount	Count	The count of session start events ingested by the app monitor. In other words, the number of new sessions started.
SessionDuration	Milliseconds	The duration of a session. These are calculated based on the time between first and last events in a session.
TimeOnPage	Milliseconds	The duration of a page view. These are calculate d based on the time until next page view, except for the final page in a session where it's the time between first and last events on that page.
WebVitalsCumulativeLayoutShift	None	Tracks the value of the cumulative layout shift events.
WebVitalsFirstInputDelay	Milliseconds	Tracks the value of the first input delay events.

Metric	Unit	Description
`WebVitalsLargestContentfulPaint`	Milliseconds	Tracks the value of the largest contentful paint events.
`WebVitalsInteractionToNextPaint`	Milliseconds	Tracks the value of the interaction to next paint events.

Custom metrics and extended metrics that you can send to CloudWatch and CloudWatch Evidently

By default, RUM app monitors send metrics to CloudWatch. These default metrics and dimensions are listed in CloudWatch metrics that you can collect with CloudWatch RUM.

You can also set up an app monitor to export metric. The app monitor can send extended metrics, custom metrics, or both. It can send them to CloudWatch or to CloudWatch Evidently, or to both.

- **Custom metrics**– Custom metrics are metrics that you define. With custom metrics, you can use any metric name and namespace. To derive the metrics, you can use any custom events, built-in events, custom attributes, or default attributes.

  You can send custom metrics to both CloudWatch and CloudWatch Evidently.
- **Extended metrics**– Lets you send the default CloudWatch RUM metrics to CloudWatch Evidently to be used in Evidently experiments. You can also send any of the default CloudWatch RUM metrics to CloudWatch with additional dimensions. This way, these metrics can give you a more fine-grained view.

Topics

- Custom metrics
- Extended metrics

Custom metrics

To send custom metrics, you must use the Amazon APIs or Amazon CLI instead of the console. For more information about using the Amazon APIs, see PutRumMetricsDestination and BatchCreateRumMetricDefinitions.

The maximum number of extended metric and custom metric definitions that one destination can contain is 2000. For each custom metric or extended metric that you send to each destination, each combination of dimension name and dimension value counts toward this limit. You are not charged for custom metrics derived from any kind of events or attributes of CloudWatch RUM.

The following example shows how to create a custom metric derived from a custom event. Here is the example custom event that is used:

```
cwr('recordEvent', {
    type: 'my_custom_event',
    data: {
        location: 'IAD',
        current_url: 'amazonaws.com',
        user_interaction: {
            interaction_1 : "click",
            interaction_2 : "scroll"
        },
        visit_count:10
    }
})
```

Given this custom event, you can create a custom metric that counts the number of visits to the `amazonaws.com` URL from Chrome browsers. The following definition creates a metric named `AmazonVisitsCount` in your account, in the `RUM/CustomMetrics/PageVisits` namespace.

```
{
    "AppMonitorName":"customer-appMonitor-name",
    "Destination":"CloudWatch",
    "MetricDefinitions":[
        {
            "Name":"AmazonVisitsCount",
            "Namespace":"PageVisit",
            "ValueKey":"event_details.visit_count",
            "UnitLabel":"Count",
            "DimensionKeys":{
                "event_details.current_url": "URL"
```

```
            },
            "EventPattern":"{\"metadata\":{\"browserName\":[\"Chrome\"]},\"event_type
\":[\"my_custom_event\"],\"event_details\": {\"current_url\": [\"amazonaws.com\"]}}"
        }
    ]
}
```

Extended metrics

If you set up extended metrics, you can do one or both of the following:

- Send default CloudWatch RUM metrics to CloudWatch Evidently to be used in Evidently experiments. Only the **PerformanceNavigationDuration**, **PerformanceResourceDuration**, **WebVitalsCumulativeLayoutShift**, **WebVitalsFirstInputDelay**, and **WebVitalsLargestContentfulPaint** metrics can be sent to Evidently.
- Send any of the default CloudWatch RUM metrics to CloudWatch with additional dimensions so that the metrics give you a more fine-grained view. For example, you can see metrics specific to a certain browser that's used by your users, or metrics for users in a specific geolocation.

For more information about the default CloudWatch RUM metrics, see [CloudWatch metrics that you can collect with CloudWatch RUM](#).

The maximum number of extended metric and custom metric definitions that one destination can contain is 2000. For each extended or custom metric that you send to each destination, each combination of dimension name and dimension value counts as an extended metric for this limit.

When you send extended metrics to CloudWatch, you can use the CloudWatch RUM console to create CloudWatch alarms on them.

You are not charged for extended metrics that are created for the default metrics of CloudWatch RUM.

The following dimensions are supported for extended metrics for all the metric names that app monitors can send. These metric names are listed in [CloudWatch metrics that you can collect with CloudWatch RUM](#).

- `BrowserName`

  Example dimension values: `Chrome`, `Firefox`, `Chrome Headless`

- `CountryCode` This uses the ISO-3166 format, with two-letter codes.

  Example dimension values: `US`, `JP`, `DE`
- `DeviceType`

  Example dimension values: `desktop`, `mobile`, `tablet`, `embedded`
- `FileType`

  Example dimension values: `Image`, `Stylesheet`
- `OSName`

  Example dimension values: `Linux`, `Windows, iOS`, `Android`
- `PageId`

Set up extended metrics using the console

To use the console to send extended metrics to CloudWatch, use the following steps.

To send extended metrics to CloudWatch Evidently, you must use the Amazon APIs or Amazon CLI instead of the console. For information about using the Amazon APIs to send extended metrics to either CloudWatch or Evidently, see PutRumMetricsDestination and BatchCreateRumMetricDefinitions.

To use the console to set up an app monitor and send RUM extended metrics to CloudWatch

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Application Signals**, **RUM**.
3. Choose **List view** and then choose the name of the app monitor that is to send the metrics.
4. Choose the **Configuration** tab and then choose **RUM extended metrics**.
5. Choose **Send metrics**.
6. Select one or more metric names to send with additional dimensions.
7. Select one or more factors to use as dimensions for these metrics. As you make your choices, the number of extended metrics that your choices create is displayed in **Number of extended metrics**.

   This number is calculated by multiplying the number of chosen metric names by the number of different dimensions that you create.

a. To send a metric with page ID as a dimension, choose **Browse for page ID** and then select the page IDs to use.

b. To send a metric with device type as a dimension, choose either **Desktop devices** or **Mobile and tablets**.

c. To send a metric with operating system as a dimension, select one or more operating systems under **Operating system**.

d. To send a metric with browser type as a dimension, select one or more browsers under **Browsers**.

e. To send a metric with geolocation as a dimension, select one or more locations under **Locations**.

   Only the locations where this app monitor has reported metrics from will appear in the list to choose from.

8. When you are finished with your choices, choose **Send metrics**.

9. (Optional) In the **Extended metrics** list, to create an alarm that watches one of the metrics, choose **Create alarm** in that metric's row.

   For general information about CloudWatch alarms, see Using Amazon CloudWatch alarms. For a tutorial for setting an alarm on a CloudWatch RUM extended metric, see Tutorial: create an extended metric and alarm it.

Stop sending extended metrics

To use the console to stop sending extended metrics

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Application Signals**, **RUM**.
3. Choose **List view** and then choose the name of the app monitor that is to send the metrics.
4. Choose the **Configuration** tab and then choose **RUM extended metrics**.
5. Select one or more metric name and dimension combinations to stop sending. Then choose **Actions**, **Delete**.

Tutorial: create an extended metric and alarm it

This tutorial demonstrates how to set up an extended metric to be sent to CloudWatch, and then how to set an alarm on that metric. In this tutorial, you create a metric that tracks JavaScript errors on the Chrome browser.

To set up this extended metric and set an alarm on it

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Application Signals**, **RUM**.
3. Choose **List view** and then choose the name of the app monitor that is to send the metric.
4. Choose the **Configuration** tab and then choose **RUM extended metrics**.
5. Choose **Send metrics**.
6. Select **JSErrorCount**.
7. Under **Browsers**, select **Chrome**.

   This combination of **JSErrorCount** and **Chrome** will send one extended metric to CloudWatch. The metric counts JavaScript errors only for user sessions that use the Chrome browser. The metric name will be **JsErrorCount** and the dimension name will be **Browser**.
8. Choose **Send metrics**.
9. In the **Extended metrics** list, choose **Create alarm** in the row that displays **JsErrorCount** under **Name** and displays **Chrome** under **BrowserName**.
10. Under **Specify metric and conditions**, confirm that the **Metric name** and **BrowserName** fields are pre-filled with the correct values.
11. For **Statistic**, select the statistic that you want to use for the alarm. **Average** is a good choice for this type of counting metric.
12. For **Period**, select **5 minutes**.
13. Under **Conditions**, do the following:

    - Choose **Static**.
    - Choose **Greater** to specify that the alarm should go into ALARM state when the number of errors is higher than the threshold you are about to specify.
    - Under **than...**, enter the number for the alarm threshold. The alarm goes into ALARM state when the number of errors over a 5-minute period exceeds this number.

14. (Optional) By default, the alarm goes into ALARM state as soon as the number of errors exceeds the threshold number you set during a 5-minute period. You can optionally change this so that the alarm goes into ALARM state only if this number is exceeded for more than one 5-minute period.

    To do so, choose **Additional configuration** and then for **Datapoints to alarm**, specify how many 5-minute periods need to have the error number over the threshold to trigger the alarm. For example, you can select 2 out of 2 to have the alarm trigger only when two consecutive 5-minute periods are over the threshold, or 2 out of 3 to have the alarm trigger if any two of three consecutive 5-minute periods are over the threshold.

    For more information about this type of alarm evaluation, see Evaluating an alarm.
15. Choose **Next**.
16. For **Configure actions**, specify what should happen when the alarm goes into alarm state. To receive a notification with Amazon SNS, do the following:
    - Choose **Add notification**.
    - Choose **In alarm**.
    - Either select an existing SNS topic or create a new one. If you create a new one, specify a name for it and add at least one email address to it.
17. Choose **Next**.
18. Enter a name and optional description for the alarm, and choose **Next**.
19. Review the details and choose **Create alarm**.

Data protection and data privacy with CloudWatch RUM

The Amazon shared responsibility model applies to data protection and data privacy in Amazon CloudWatch RUM. As described in this model, Amazon is responsible for protecting the global infrastructure that runs all of the Amazon cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. For more information about data privacy, see the Data Privacy FAQ. For information about data protection in Europe, see The Amazon Shared Responsibility Model and GDPR blog post on the Amazon Security Blog. For more resources about complying with GDPR requirements, see the General Data Protection Regulation (GDPR) Center.

Amazon CloudWatch RUM generates a code snippet for you to embed into your website or web application code, based on your input of end user data that you want to collect. The web client, downloaded and configured by the code snippet, uses cookies (or similar technologies) to help you collect end user data. The use of cookies (or similar technologies) is subject to data privacy regulations in certain jurisdictions. Before using Amazon CloudWatch RUM, we strongly recommend that you assess your compliance obligations under applicable law, including any applicable legal requirements to provide legally adequate privacy notices and obtain any necessary consents for the use of cookies and the processing (including collection) of end user data. For more information about how the web client uses cookies (or similar technologies) and what end-user data the web client collects, see Information collected by the CloudWatch RUM web client and CloudWatch RUM web client cookies (or similar technologies).

We strongly recommend that you never put sensitive identifying information, such as your end users' account numbers, email addresses, or other personal information, into free-form fields. Any data that you enter into Amazon CloudWatch RUM or other services might be included in diagnostic logs.

CloudWatch RUM web client cookies (or similar technologies)

The CloudWatch RUM web client collects certain data about user sessions by default. You can choose to enable cookies to have the web client collect a user ID and session ID that persist across page loads. The user ID is randomly generated by RUM.

If these cookies are enabled, RUM is able to display the following types of data when you view the RUM dashboard for this app monitor.

- Aggregated data based on user IDs, such as number of unique users and the number of different users who experienced an error.
- Aggregated data based on session IDs, such as number of sessions and the number of sessions that experienced an error.
- The *user journey*, which is the sequence of pages that each sampled user session includes.

Important

If you do not enable these cookies (or similar technologies), the web client still records certain information about end user sessions such as browser type/version, operating system type/version, device type, and so on. These are collected to provide aggregated page-specific insights, such as web vitals, page views, and pages that experienced errors. For more information about the data recorded, see Information collected by the CloudWatch RUM web client.

Information collected by the CloudWatch RUM web client

This section documents the **PutRumEvents** schema, which defines the structure of the data that you can collect from user sessions using CloudWatch RUM.

A **PutRumEvents** request sends a data structure with the following fields to CloudWatch RUM.

- The ID of this batch of RUM events
- App monitor details, which includes the following:
  - App monitor ID
  - Monitored application version
- User details, which includes the following. **This is collected only if the app monitor has cookies enabled.**
  - A user ID generated by the web client
  - Session ID
- The array of RUM events in this batch.

RUM event schema

The structure of each RUM event includes the following fields.

- The ID of the event
- A timestamp
- The event type
- The user agent
- Metadata
- RUM event details

RUM event metadata

The metadata includes page metadata, user agent metadata, geolocation metadata, and domain metadata.

Page metadata

The page metadata includes the following:

- Page ID
- Page title
- Parent page ID. – **This is collected only if the app monitor has cookies enabled.**
- Interaction depth – **This is collected only if the app monitor has cookies enabled.**
- Page tags – You can add tags to page events to group pages together. For more information, see Use page groups.

User agent metadata

The user agent metadata includes the following:

- Browser language
- Browser name
- Browser version
- Operating system name
- Operating system version
- Device type
- Platform type

Geolocation metadata

The geolocation metadata includes the following:

- Country code
- Subdivision code

Domain metadata

The domain metadata includes the URL domain.

RUM event details

The details of an event follow one of the following type of schemas, depending on the event type.

Session start event

This event contains no fields. **This is collected only if the app monitor has cookies enabled.**

Page view schema

A **Page view** event contains the following properties. You can deactivate page view collection by configuring the web client. For more information, see the CloudWatch RUM web client documentation.

Name	Type	Description
Page ID	String	An ID that uniquely represents this page within the application. By default, this is the URL path.
Parent page ID	String	The ID of the page that the user was on when they navigated to the current page. **This is collected only if the app monitor has cookies enabled.**
Interaction depth	String	**This is collected only if the app monitor has cookies enabled.**

JavaScript error schema

JavaScript error events generated by the agent contain the following properties. The web client collects these events only if you selected to collect the errors telemetry.

Name	Type	Description
Error type	String	The error's name, if one exists. For more information, see Error.prototype.name.

Name	Type	Description
		Some browsers might not support error types.
Error message	String	The error's message. For more information, see Error.prototype.message. If the error field does not exist, this is the message of the error event. For more information, see ErrorEvent. Error messages might not be consistent across different browsers.
Stack trace	String	The error's stack trace, if one exists, truncated to 150 characters. For more information, see Error.prototype.stack. Some browsers might not support stack traces.

DOM event schema

Document object model (DOM) events generated by the agent contain the following properties. These events are not collected by default. They are collected only if you activate the interactions telemetry. For more information, see the CloudWatch RUM web client documentation.

Name	Type	Description
Event	String	The type of DOM event, such as click, scroll, or hover. For more information, see Event reference.
Element	String	The DOM element type
Element ID	String	If the element that generated the event has an ID, this property stores that ID. For more information, see Element.id.
CSSLocator	String	The CSS locator used to identify the DOM element.
InteractionId	String	A unique ID for the interaction between the user and the UI.

Navigation event schema

Navigation events are collected only if the app monitor has performance telemetry activated.

Navigation events use Navigation timing Level 1 and Navigation timing Level 2 APIs. Level 2 APIs are not supported on all browsers, so these newer fields are optional.

> **Note**
>
> Timestamp metrics are based on DOMHighResTimestamp. With Level 2 APIs, all timings are by default relative to the `startTime`. But for Level 1, the `navigationStart` metric is subtracted from timestamp metrics to obtain relative values. All timestamp values are in milliseconds.

Navigation events contain the following properties.

Name	Type	Description	Notes
initiatorType	String	Represents the type of resource that initiated the performance event.	**Value:** "navigation" **Level 1:** "navigation" **Level 2:** entryData .initiatorType
navigationType	String	Represents the type of navigation. This attribute is not required.	**Value:** The value must be one of the following: • `navigate` is a navigation started by choosing

Name	Type	Description	Notes
			a link, entering a URL in a browser's address bar, form submission, or initializing through a script operation other than `reload` or `back_forward` . • `reload` is a navigation through the browser's reload operation or `location.reload()` . • `back_forward` is a navigation through the browser's history

Name	Type	Description	Notes
			traversal operation. • `prerender` is a navigation initiated by a prerender hint. For more information, see [Prerender]().
startTime	Number	Indicates when the event is triggered.	**Value:** 0 **Level 1:** entryData .navigationStart - entryData .navigationStart **Level 2:** entryData .startTime

Name	Type	Description	Notes
unloadEventStart	Number	Indicates the time when the previous document in the window began to unload after the `unload` event was thrown.	**Value:** If there is no previous document or if the previous document or one of the needed redirects is not of the same origin, the value returned is 0. **Level 1:** `entryData.unloadEventStart > 0 ? entryData.unloadEventStart - entryData.navigationStart : 0` **Level 2:** entryData.unloadEventStart

Name	Type	Description	Notes
promptForUnload	Number	The time taken to unload the document. In other words, the time between `unloadEventStart` and `unloadEventEnd`. `UnloadEventEnd` represents the moment in milliseconds when the unload event handler finishes.	**Value:** If there is no previous document or if the previous document or one of the needed redirects is not of the same origin, the value returned is 0. **Level 1:** entryData.unloadEventEnd - entryData.unloadEventStart **Level 2:** entryData.unloadEventEnd - entryData.unloadEventStart

Name	Type	Description	Notes
redirectCount	Number	A number representing the number of redirects since the last non-redirect navigation under the current browsing context. This attribute is not required.	**Value:** If there is no redirect or if there is any redirect that is not of the same origin as the destination document, the value returned is 0. **Level 1:** Not available **Level 2:** entryData.redirectCount

Name	Type	Description	Notes
redirectStart	Number	The time when the first HTTP redirect starts.	**Value:** If there is no redirect or if there is any redirect that is not of the same origin as the destination document, the value returned is 0. **Level 1:** `entryData.redirectStart > 0 ? entryData.redirectStart - entryData.navigationStart : 0` **Level 2:** entryData .redirectStart

Name	Type	Description	Notes
redirectTime	Number	The time taken for the HTTP redirect. This is the difference between `redirectStart` and `redirectEnd` .	**Level 1:** : entryData .redirectEnd - entryData .redirectStart **Level 2:** : entryData .redirectEnd - entryData .redirectStart

Name	Type	Description	Notes
workerStart	Number	This is a property of the `PerformanceResourceTiming` interface. It marks the beginning of worker thread operation. This attribute is not required.	**Value:** If a Service Worker thread is already running, or immediately before starting the Service Worker thread, this property returns the time immediately before dispatching `FetchEvent`. It returns 0 if the resource is not intercepted by a Service Worker. **Level 1:** Not available **Level 2:** entryData.workerStart

Name	Type	Description	Notes
workerTime	Number	If the resource is intercepted by a Service Worker, this returns the time required for worker thread operation. This attribute is not required.	**Level 1:** Not available **Level 2:** `entryData.workerStart > 0 ? entryData.fetchStart - entryData.workerStart : 0`
fetchStart	Number	The time when the browser is ready to fetch the document using an HTTP request. This is before checking any application cache.	**Level 1:** `: entryData.fetchStart > 0 ? entryData.fetchStart - entryData.navigationStart : 0` **Level 2:** entryData .fetchStart

Name	Type	Description	Notes
domainLookupStart	Number	The time when the domain lookup starts.	**Value:** If a persistent connection is used or if the information is stored in a cache or local resource, the value will be the same as `fetchStart` . **Level 1:** `entryData.domainLookupStart > 0 ? entryData.domainLookupStart - entryData.navigationStart : 0` **Level 2:** entryData.domainLookupStart

Name	Type	Description	Notes
dns	Number	The time required for domain lookup.	**Value:** If the resources and DNS records are cached, the expected value is 0. **Level 1:** entryData .domainLo okupEnd - entryData .domainLo okupStart **Level 2:** entryData .domainLo okupEnd - entryData .domainLo okupStart
nextHopPr otocol	String	A string representing the network protocol used to fetch the resource. This attribute is not required.	**Level 1:** Not available **Level 2:** entryData .nextHopP rotocol

Name	Type	Description	Notes
connectStart	Number	The time immediately before the user agent starts establishing the connection to the server to retrieve the document.	**Value:** If an RFC2616 persistent connection is used, or if the current document is retrieved from relevant applicati on caches or local resources, this attribute returns the value of `domainLoo kupEnd` . **Level 1:** `entryData .connectS tart > 0 ? entryData .connectS tart - entryData .navigati onStart : 0` **Level 2:** entryData .connectStart

Name	Type	Description	Notes
connect	Number	Measures the time required to establish the transport connections or to perform SSL authentication. It also includes the blocked time that is taken when there are too many concurrent requests issued by the browser.	**Level 1:** entryData .connectEnd - entryData .connectStart **Level 2:** entryData .connectEnd - entryData .connectStart
secureCon nectionStart	Number	If the URL scheme of the current page is "https", this attribute returns the time immediately before the user agent starts the handshake process to secure the current connection. It returns 0 if HTTPS is not used. For more information about URL schemes, see [URL representation]().	**Formula:** entryData .secureCo nnectionS tart

Name	Type	Description	Notes
tlsTime	Number	The time taken to complete an SSL handshake.	**Level 1:** `entryData.secureConnectionStart > 0 ? entryData.connectEnd - entryData.secureConnectionStart : 0` **Level 2:** `entryData.secureConnectionStart > 0 ? entryData.connectEnd - entryData.secureConnectionStart : 0`

Name	Type	Description	Notes
requestStart	Number	The time immediately before the user agent starts requesting the resource from the server, or from relevant application caches, or from local resources.	**Level 1:** `: entryData.requestStart > 0 ? entryData.requestStart - entryData.navigationStart : 0` **Level 2:** entryData .requestStart
timeToFirstByte	Number	The time taken to receive the first byte of information after a request is made. This time is relative to the `startTime` .	**Level 1:** entryData .response Start - entryData .requestStart **Level 2:** entryData .response Start - entryData .requestStart

Name	Type	Description	Notes
responseStart	Number	The time immediately after the user agent's HTTP parser receives the first byte of the response from the relevant application caches, or from local resources, or from the server.	**Level 1:** `entryData.responseStart > 0 ? entryData.responseStart - entryData.navigationStart : 0` **Level 2:** entryData.responseStart

Name	Type	Description	Notes
responseTime	String	The time taken to receive a complete response in the form of bytes from the relevant application caches, or from local resources, or from the server.	**Level 1:** `entryData.responseStart > 0 ? entryData.responseEnd - entryData.responseStart : 0` **Level 2:** `entryData.responseStart > 0 ? entryData.responseEnd - entryData.responseStart : 0`

Name	Type	Description	Notes
domInteractive	Number	The time when the parser finished its work on the main document, and the HTML DOM is constructed. At this time, its `Document.readyState` changes to "interactive" and the corresponding `readystatechange` event is thrown.	**Level 1:** `entryData.domInteractive > 0 ? entryData.domInteractive - entryData.navigationStart : 0` **Level 2:** entryData.domInteractive

Name	Type	Description	Notes
domContentLoadedEventStart	Number	Represents the time value equal to the time immediately before the user agent fires the DOMContentLoaded event at the current document. TheDOMContentLoaded event fires when the initial HTML document has been completely loaded and parsed. At this time, the main HTML document has finished parsing, the browser begins constructing the render tree, and subresources still have to be loaded. This does not wait for style sheets, images, and subframes to finish loading.	**Level 1:** `entryData.domContentLoadedEventStart > 0 ? entryData.domContentLoadedEventStart - entryData.navigationStart : 0` **Level 2:** entryData.domContentLoadedEventStart

Name	Type	Description	Notes
domContentLoaded	Number	This start and end time of render tree construction is marked by the `domContentLoadedEventStart` and `domContentLoadedEventEnd`. It lets CloudWatch RUM track execution. This property is the difference between `domContentLoadedStart` and `domContentLoadedEnd`. During this time, DOM and CSSOM are ready. This property waits on script execution, except for asynchronous and dynamically created scripts. If the scripts depend on style sheets, `domContentLoaded` waits on the style sheets, too. It does not wait on images. **Note** The actual values of `domContentLoadedStart` and `domContentLoadedEnd` approximate to `domContentLoaded` in Google Chrome's Network panel. It indicates HTML DOM + CSSOM render tree construction time from the beginning of the page loading process. In the case of navigation metrics, the `domContentLoaded` value represents the difference between start and end values, which is the time required for downloading subresources and render-tree construction only.	**Level 2:** entryData.domContentLoadedEventEnd - entryData.domContentLoadedEventStart **Level 2:** entryData.domContentLoadedEventEnd - entryData.domContentLoadedEventStart

Name	Type	Description	Notes
domComplete	Number	The time immediately before the browser sets the current document readiness of the current document to complete. At this point, the loading of subresources, such as images, is complete. This includes the time taken for downloading blocking content such as CSS and synchronous JavaScript. This approximates to `loadTime` in Google Chrome's Network panel.	**Level 1:** `entryData.domComplete > 0 ? entryData.domComplete - entryData.navigationStart : 0` **Level 2:** entryData.domComplete
domProcessingTime	Number	The total time between the response and the load event start.	**Level 1:** entryData.loadEventStart - entryData.responseEnd **Level 2:** entryData.loadEventStart - entryData.responseEnd

Name	Type	Description	Notes
loadEvent Start	Number	The time immediately before the `load` event of the current document is fired.	**Level 1:** `entryData.loadEventStart > 0 ? entryData.loadEventStart - entryData.navigationStart : 0` **Level 2:** entryData.loadEventStart
loadEvent Time	Number	The difference between `loadEventStart` and `loadEventEnd` . Additional functions or logic waiting for this load event will be fired during this time.	**Level 1:** entryData.loadEventEnd - entryData.loadEventStart **Level 2:** entryData.loadEventEnd - entryData.loadEventStart

Name	Type	Description	Notes
duration	String	Duration is the total page load time. It records the timing for downloading the main page and all of its synchronous subresources, and also for rendering the page. Asynchronous resources such as scripts continue to download later. This is the difference between the `loadEventEnd` and `startTime` properties.	**Level 1:** entryData .loadEven tEnd - entryData .navigati onStart **Level 2:** entryData .duration
headerSize	Number	Returns the difference between `transferSize` and `encodedBodySize`. This attribute is not required.	**Level 1:** Not available **Level 2:** entryData .transferSize - entryData .encodedB odySize **Level 2:** entryData .transferSize - entryData .encodedB odySize

Name	Type	Description	Notes
compressionRatio	Number	The ratio of `encodedBodySize` and `decodedBodySize`. The value of `encodedBodySize` is the compressed size of the resource excluding the HTTP headers. The value of `decodedBodySize` is the decompressed size of the resource excluding the HTTP headers. This attribute is not required.	**Level 1:** Not available. **Level 2:** `entryData.encodedBodySize > 0 ? entryData.decodedBodySize / entryData.encodedBodySize : 0`
navigationTimingLevel	Number	The navigation timing API version.	**Value:** 1 or 2

Resource event schema

Resource events are collected only if the app monitor has performance telemetry activated.

Timestamp metrics are based on [The DOMHighResTimeStamp typedef](). With Level 2 APIs, by default all timings are relative to the `startTime`. But for Level 1 APIs, the `navigationStart` metric is subtracted from timestamp metrics to obtain relative values. All timestamp values are in milliseconds.

Resource events generated by the agent contain the following properties.

Name	Type	Description	Notes
targetUrl	String	Returns the resource's URL.	**Formula:** entryData .name
initiatorType	String	Represents the type of resource that initiated the performance resource event.	**Value:** "resource" **Formula:** entryData .initiatorType
duration	String	Returns the difference between the `responseEnd` and `startTime` properties. This attribute is not required.	**Formula:** entryData .duration
transferSize	Number	Returns the size (in octets) of the fetched resource, including the response header fields and the response payload body. This attribute is not required.	**Formula:** entryData .transferSize
fileType	String	Extensions derived from the target URL pattern.	

Largest contentful paint event schema

Largest contentful paint events contain the following properties.

These events are collected only if the app monitor has performance telemetry activated.

Name	Description		
Value	For more information, see Web Vitals.		

First input delay event

First input delay events contain the following properties.

These events are collected only if the app monitor has performance telemetry activated.

Name	Description		
Value	For more information, see Web Vitals.		

Cumulative layout shift event

Cumulative layout shift events contain the following properties.

These events are collected only if the app monitor has performance telemetry activated.

Name	Description		
Value	For more information, see Web Vitals.		

HTTP event

HTTP events can contain the following properties. It will contain either a `Response` field or an `Error` field, but not both.

These events are collected only if the app monitor has HTTP telemetry activated.

Name	Description
Request	The request field includes the following: • The `Method` field, which can have values such as GET, POST, and so on.

Name	Description
	• The URL
Response	The response field includes the following: • Status, such as 2xx, 4xx, or 5xx • Status text
Error	The error field includes the following: • Type • Message • File name • Line number • Column number • Stack trace

X-Ray trace event schema

These events are collected only if the app monitor has X-Ray tracing activated.

For information about X-Ray trace event schemas, see Amazon X-Ray segment documents.

Route change timing for single-page applications

In a traditional multi-page application, when a user requests for new content to be loaded, the user is actually requesting a new HTML page from the server. As a result, the CloudWatch RUM web client captures the load times using the regular performance API metrics.

However, single-page web applications use JavaScript and Ajax to update the interface without loading a new page from the server. Single-page updates are not recorded by the browser timing API, but instead use route change timing.

CloudWatch RUM supports the monitoring of both full page loads from the server and single-page updates, with the following differences:

- For route change timing, there are no browser-provided metrics such as `tlsTime`, `timeToFirstByte`, and so on.

- For route change timing, the `initiatorType` field will be `route_change`.

The CloudWatch RUM web client listens to user interactions that may lead to a route change, and when such a user interaction is recorded, the web client records a timestamp. Then route change timing will begin if both of the following are true:

- A browser history API (except browser forward and back buttons) was used to perform the route change.
- The difference between the time of route change detection and latest user interaction timestamp is less than 1000 ms. This avoids data skew.

Then, once route change timing begins, that timing completes if there are no ongoing AJAX requests and DOM mutations. Then the timestamp of the latest completed activity will be used as the completion timestamp.

Route change timing will time out if there are ongoing AJAX requests or DOM mutations for more than 10 seconds (by default). In this case, the CloudWatch RUM web client will no longer record timing for this route change.

As a result, the duration of a route change event is calculated as the following:

```
(time of latest completed activity) - (latest user interaction timestamp)
```

Manage your applications that use CloudWatch RUM

Use the steps in these sections to manage your applications' use of CloudWatch RUM.

Topics

- How do I find a code snippet that I've already generated?
- Editing your CloudWatch RUM app monitor settings
- Stopping using CloudWatch RUM or deleting an app monitor

How do I find a code snippet that I've already generated?

To find a CloudWatch RUM code snippet that you've already generated for an application, follow these steps.

To find a code snippet that you've already generated

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Application Signals**, **RUM**.
3. Choose **List view**.
4. Next to the name of the app monitor, choose **View JavaScript**.
5. In the **JavaScript Snippet** pane, choose **Copy to clipboard**.

Editing your CloudWatch RUM app monitor settings

To change an app monitor's settings, follow these steps. You can change any settings except the app monitor name.

To edit how your application uses CloudWatch RUM

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Application Signals**, **RUM**.
3. Choose **List view**.
4. Choose the button next to the name of the application, and then choose **Actions**, **Edit**.
5. Change any settings except the application name. For more information about the settings, see Creating a CloudWatch RUM app monitor.
6. When finished, choose **Save**.

   Changing the settings changes the code snippet. You must now paste the updated code snippet into your application.
7. After the JavaScript code snippet is created, choose **Copy to clipboard** or **Download**, and then choose **Done**.

   To start monitoring with the new settings, you insert the code snippet into your application. Insert the code snippet inside the `<head>` element of your application, before the `<body>` element or any other `<script>` tags.

Stopping using CloudWatch RUM or deleting an app monitor

To stop using CloudWatch RUM with an application, remove the code snippet that RUM generated from your application's code.

To delete a RUM app monitor, follow these steps.

To delete an app monitor

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Application Signals**, **RUM**.
3. Choose **List view**.
4. Choose the button next to the name of the application, and then choose **Actions**, **Delete**.
5. In the confirmation box, enter **`Delete`** and then choose **Delete**.
6. If you haven't done so already, delete the CloudWatch RUM code snippet from your application's code.

CloudWatch RUM quotas

CloudWatch RUM has the following quotas.

Resource	Default quota
App monitors	20 per account You can request a quota increase.
RUM ingestion rate	50 **PutRumEvents** requests per second (TPS). You can request a quota increase.

Troubleshooting CloudWatch RUM

This section contains tips to help you troubleshoot CloudWatch RUM.

There is no data for my application

First, make sure that the code snippet has been correctly inserted into your application. For more information, see Inserting the CloudWatch app monitor code snippet into your application.

If that is not the issue, then maybe there has been no traffic to your application yet. Generate some traffic by accessing your application the same way that a user would.

Data has stopped being recorded for my application

Your application might have been updated and now no longer contains a CloudWatch RUM code snippet. Check your application code.

Another possibility is that someone may have updated the code snippet but then didn't insert the updated snippet into the application. Find the current correct code snippet by following the directions in How do I find a code snippet that I've already generated? and compare it to the code snippet that is pasted into your application.

Perform launches and A/B experiments with CloudWatch Evidently

> **Important**
>
> End of support notice: On October 16, 2025, Amazon will discontinue support for CloudWatch Evidently. After October 16, 2025, you will no longer be able to access the Evidently console or Evidently resources.

You can use Amazon CloudWatch Evidently to safely validate new features by serving them to a specified percentage of your users while you roll out the feature. You can monitor the performance of the new feature to help you decide when to ramp up traffic to your users. This helps you reduce risk and identify unintended consequences before you fully launch the feature.

You can also conduct A/B experiments to make feature design decisions based on evidence and data. An experiment can test as many as five variations at once. Evidently collects experiment data and analyzes it using statistical methods. It also provides clear recommendations about which variations perform better. You can test both user-facing features and backend features.

Evidently pricing

Evidently charges your account based on Evidently events and Evidently analysis units. Evidently events include both data events such as clicks and page views, and assignment events that determine the feature variation to serve to a user.

Evidently analysis units are generated from Evidently events, based on rules that you have created in Evidently. Analysis units are the number of rule matches on events. For example, a user click event might produce a single Evidently analysis unit, a click count. Another example is a user checkout event that might produce two Evidently analysis units, checkout value and the number of items in cart. For more information about pricing, see Amazon CloudWatch Pricing.

CloudWatch Evidently is currently available in the following Regions:

- US East (Ohio)
- US East (N. Virginia)
- US West (Oregon)
- Asia Pacific (Singapore)
- Asia Pacific (Sydney)
- Asia Pacific (Tokyo)
- Europe (Frankfurt)
- Europe (Ireland)
- Europe (Stockholm)

Topics

IAM policies to use Evidently

> **Important**
>
> End of support notice: On October 16, 2025, Amazon will discontinue support for CloudWatch Evidently. After October 16, 2025, you will no longer be able to access the Evidently console or Evidently resources.

To fully manage CloudWatch Evidently, you must be signed in as an IAM user or role that has the following permissions:

- The **AmazonCloudWatchEvidentlyFullAccess** policy
- The **ResourceGroupsandTagEditorReadOnlyAccess** policy

Additionally, to be able to create a project that stores evaluation events in Amazon S3 or CloudWatch Logs, you need the following permissions:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketPolicy",
                "s3:PutBucketPolicy",
                "s3:GetObject",
                "s3:ListBucket"
            ],
            "Resource": "arn:aws:s3:::*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogDelivery",
                "logs:DeleteLogDelivery",
                "logs:DescribeResourcePolicies",
                "logs:PutResourcePolicy"
            ],
            "Resource": [
```

```
                "*"
            ]
        }
    ]
}
```

Additional permissions for CloudWatch RUM integration

Additionally, if you intend to manage Evidently launches or experiments that integrate with Amazon CloudWatch RUM and use CloudWatch RUM metrics for monitoring, you need the **AmazonCloudWatchRUMFullAccess** policy. To create an IAM role to give the CloudWatch RUM web client permission to send data to CloudWatch RUM, you need the following permissions:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iam:CreateRole",
                "iam:CreatePolicy",
                "iam:AttachRolePolicy"
            ],
            "Resource": [
                "arn:aws:iam::*:role/service-role/CloudWatchRUMEvidentlyRole-*",
                "arn:aws:iam::*:policy/service-role/CloudWatchRUMEvidentlyPolicy-*"
            ]
        }
    ]
}
```

Permissions for read-only access to Evidently

For other users who need to view Evidently data but don't need to create Evidently resources, you can grant the **AmazonCloudWatchEvidentlyReadOnlyAccess** policy.

Create projects, features, launches, and experiments

> **Important**
>
> End of support notice: On October 16, 2025, Amazon will discontinue support for CloudWatch Evidently. After October 16, 2025, you will no longer be able to access the Evidently console or Evidently resources.

To get started with CloudWatch Evidently, for either a feature launch or an A/B experiment, you first create a *project*. A project is a logical grouping of resources. Within the project, you create *features* that have variations that you want to test or launch. You can create a feature either before you create a launch or experiment, or at the same time.

Topics

Create a new project

> **Important**
>
> End of support notice: On October 16, 2025, Amazon will discontinue support for CloudWatch Evidently. After October 16, 2025, you will no longer be able to access the Evidently console or Evidently resources.

Use these steps to set up a new CloudWatch Evidently project.

To create a new CloudWatch Evidently project

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.

2. In the navigation pane, choose **Application Signals**, **Evidently**.
3. Choose **Create project**.
4. For **Project name**, enter a name to be used to identify this project within the CloudWatch Evidently console.

   You can optionally add a project description.
5. For **Evaluation event storage**, choose whether you want to store the evaluation events that you collect with Evidently. Even if you don't store these events, Evidently aggregates them to create metrics and other experiment data that you can view in the Evidently dashboard. For more information, see Evidently project data storage in CloudWatch.
6. For **Use client-side evaluation**, choose whether you want to enable client-side evaluation for this project. With client-side evaluation, your application can assign variations to user sessions locally instead of by calling the EvaluateFeature operation. This mitigates the latency and availability risks that come with an API call. For more information, see Use client-side evaluation - powered by Amazon AppConfig.

   To create a project with client-side evaluation, you must have the `evidently:ExportProjectAsConfiguration` permission.

   If you enable client-side evaluation, also do the following:

   a. Choose whether to use an existing Amazon AppConfig application or create a new one.
   b. Choose whether to use an existing Amazon AppConfig environment or create a new one.

      For more information about applications and environments in Amazon AppConfig, see How Amazon AppConfig works.
7. (Optional) To add tags to this project, choose **Tags**, **Add new tag**.

   Then, for **Key**, enter a name for the tag. You can add an optional value for the tag in **Value**.

   To add another tag, choose **Add new tag** again.

   For more information, see Tagging Amazon Resources.
8. Choose **Create project**.

Use client-side evaluation - powered by Amazon AppConfig

Important

End of support notice: On October 16, 2025, Amazon will discontinue support for CloudWatch Evidently. After October 16, 2025, you will no longer be able to access the Evidently console or Evidently resources.

You can use *client-side evaluation - powered by Amazon AppConfig (client-side evaluation)* in a project, which lets your application assign variations to user sessions locally instead of assigning variations by calling the EvaluateFeature operation. This mitigates the latency and availability risks that come with an API call.

To use client-side evaluation, attach the Amazon AppConfig Lambda extension as a layer to your Lambda functions and configure the environment variables. The client-side evaluation runs as a side process on the local host. Then, you can call the **EvaluationFeature** and **PutProjectEvent** operations against `localhost`. The client-side evaluation process handles the variation assignment, caching, and data synchronization. For more information about Amazon AppConfig, see How Amazon AppConfig works.

When you integrate with Amazon AppConfig, you specify an Amazon AppConfig *application ID* and an Amazon AppConfig *environment ID* to Evidently. You can use the same application ID and environment ID across Evidently projects.

When you create a project with client-side evaluation enabled, Evidently creates an Amazon AppConfig configuration profile for that project. The configuration profile for each project will be different.

Client-side evaluation access control

Evidently client-side evaluation uses a different access control mechanism than the rest of Evidently does. We strongly recommend that you understand this so that you can implement the proper security measures.

With Evidently, you can create IAM policies that limit the actions a user can perform on individual resources. For example, you can create a user role that disallows a user from having the **EvaluateFeature** action. For more information about the Evidently actions that can be controlled with IAM policies, see Actions defined by Amazon CloudWatch Evidently.

The client-side evaluation model allows local evaluations of Evidently features that use project metadata. A user of a project with client-side evaluation enabled can call the **EvaluateFeature** API against a local host endpoint, and this API call does not reach Evidently and is not authenticated by the Evidently service's IAM policies. This call is successful even if the user doesn't have the IAM permission to use the **EvaluateFeature** action. However, a user still needs the **PutProjectEvents** permission for the agent to buffer the evaluation events or custom events and to offload data to Evidently asynchronously.

Additionally, a user must have the `evidently:ExportProjectAsConfiguration` permission to be able to create a project that uses client-side evaluation. This helps you control access to **EvaluateFeature** actions that are called during client-side evaluation.

If you aren't careful, the client-side evaluation security model can subvert the policies that you have set on the rest of Evidently. A user who has the `evidently:ExportProjectAsConfiguration` permission can create a project with client-side evaluation enabled, and then use the **EvaluateFeature** action for client-side evaluation with that project even if they are expressly denied the **EvaluateFeature** action in an IAM policy.

Get started with Lambda

Evidently currently supports client-side evaluation by using an Amazon Lambda environment. To get started, first decide which Amazon AppConfig application and environment to use. Choose an existing application and environment, or create new ones.

The following sample Amazon AppConfig Amazon CLI commands create an application and environment.

```
aws appconfig create-application --name YOUR_APP_NAME
```

```
aws appconfig create-environment --application-id YOUR_APP_ID --
name YOUR_ENVIRONMENT_NAME
```

Next, create an Evidently project by using these Amazon AppConfig resources. For more information, see Create a new project.

Client-side evaluation is supported in Lambda by using a Lambda layer. This is a public layer that is part of `AWS-AppConfig-Extension`, a public Amazon AppConfig extension created by the Amazon AppConfig service. For more information about Lambda layers, see Layer.

To use client-side evaluation, you must add this layer to your Lambda function and configure permissions and environment variables.

To add the Evidently client-side evaluation Lambda layer to your Lambda function and configure it

1. Create a Lambda function if you haven't already.
2. Add the client-side evaluation layer to your function. You can either specify its ARN or select it from the list of Amazon layers if you haven't already. For more information, see [Configuring functions to use layers] and [Available versions of the Amazon AppConfig Lambda extension].
3. Create an IAM policy named **EvidentlyAppConfigCachingAgentPolicy** with the following contents, and attach it to the function's execution role. For more information, see [Lambda execution role].

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "appconfig:GetLatestConfiguration",
                "appconfig:StartConfigurationSession",
                "evidently:PutProjectEvents"
            ],
            "Resource": "*"
        }
    ]
}
```

4. Add the required environment variable `AWS_APPCONFIG_EXTENSION_EVIDENTLY_CONFIGURATIONS` to your Lambda function. This environment variable specifies the mapping between the Evidently project and the Amazon AppConfig resources.

   If you are using this function for one Evidently project, set the value of the environment variable to: `applications/APP_ID/environments/ENVIRONMENT_ID/configurations/PROJECT_NAME`

If you are using this function for multiple Evidently projects, use a comma to separate the values, as in the following example: `applications/`*`APP_ID_1`*`/environments/`*`ENVIRONMENT_ID_1`*`/configurations/`*`PROJECT_NAME_1`*`,applications/`*`APP_ID_2`*`/environments/`*`ENVIRONMENT_ID_2`*`/configurations/`*`PROJECT_NAME_2`*

5. (Optional) Set other environment variables. For more information, see [Configuring the Amazon AppConfig Lambda extension](#).
6. In your application, get Evidently evaluations locally by sending `EvaluateFeature` to `localhost`.

   Python example:

```
import boto3
from botocore.config import Config

def lambda_handler(event, context):
    local_client = boto3.client(
        'evidently',
        endpoint_url="http://localhost:2772",
        config=Config(inject_host_prefix=False)
    )
    response = local_client.evaluate_feature(
        project=event['project'],
        feature=event['feature'],
        entityId=event['entityId']
    )
    print(response)
```

   Node.js example:

```
const AWS = require('aws-sdk');
const evidently = new AWS.Evidently({
    region: "us-west-2",
    endpoint: "http://localhost:2772",
    hostPrefixEnabled: false
});

exports.handler = async (event) => {

    const evaluation = await evidently.evaluateFeature({
```

```
        project: 'John_ETCProject_Aug2022',
        feature: 'Feature_IceCreamFlavors',
        entityId: 'John'
    }).promise()

    console.log(evaluation)
    const response = {
        statusCode: 200,
        body: evaluation,
    };
    return response;
};
```

Kotlin example:

```
String localhostEndpoint = "http://localhost:2772/"
public AmazonCloudWatchEvidentlyClient getEvidentlyLocalClient() {
    return AmazonCloudWatchEvidentlyClientBuilder.standard()

 .withEndpointConfiguration(AwsClientBuilder.EndpointConfiguration(localhostEndpoint,
 region))

 .withClientConfiguration(ClientConfiguration().withDisableHostPrefixInjection(true))
        .withCredentials(credentialsProvider)
        .build();
}

AmazonCloudWatchEvidentlyClient evidently = getEvidentlyLocalClient();

// EvaluateFeature via local client.
EvaluateFeatureRequest evaluateFeatureRequest = new
 EvaluateFeatureRequest().builder()
   .withProject(${YOUR_PROJECT}) //Required.
   .withFeature(${YOUR_FEATURE}) //Required.
   .withEntityId(${YOUR_ENTITY_ID}) //Required.
   .withEvaluationContext(${YOUR_EVAL_CONTEXT}) //Optional: a JSON object of
 attributes that you can optionally pass in as part of the evaluation event sent to
 Evidently.
   .build();

EvaluateFeatureResponse evaluateFeatureResponse =
 evidently.evaluateFeature(evaluateFeatureRequest);
```

```
// PutProjectEvents via local client.
PutProjectEventsRequest putProjectEventsRequest = new
 PutProjectEventsRequest().builder()
     .withData(${YOUR_DATA})
     .withTimeStamp(${YOUR_TIMESTAMP})
     .withType(${YOUR_TYPE})
     .build();

PutProjectEvents putProjectEventsResponse =
 evidently.putProjectEvents(putProjectEventsRequest);
```

Configure how often the client sends data to Evidently

To specify how often client-side evaluation sends data to Evidently, you can optionally configure two environment variables.

- `AWS_APPCONFIG_EXTENSION_EVIDENTLY_EVENT_BATCH_SIZE` specifies the number of events per project to batch before sending them to Evidently. Valid values are integers between 1 and 50, and the default is 40.
- `AWS_APPCONFIG_EXTENSION_EVIDENTLY_BATCH_COLLECTION_DURATION` specifies the duration in seconds to wait for events before sending them to Evidently. The default is 30.

Troubleshooting

Use the following information to help troubleshoot problems with using CloudWatch Evidently with client-side evaluation - powered by Amazon AppConfig.

An error occurred (BadRequestException) when calling the EvaluateFeature operation: HTTP method not supported for provided path

Your environment variables might be configured incorrectly. For example, you might have used `EVIDENTLY_CONFIGURATIONS` as the environment variable name instead of `AWS_APPCONFIG_EXTENSION_EVIDENTLY_CONFIGURATIONS`.

ResourceNotFoundException: Deployment not found

Your update to the project metadata has not been deployed to Amazon AppConfig. Check for an active deployment in the Amazon AppConfig environment that you used for client-side evaluation.

ValidationException: No Evidently configuration for project

Your `AWS_APPCONFIG_EXTENSION_EVIDENTLY_CONFIGURATIONS` environment variable might be configured with the incorrect project name.

Add a feature to a project

> **Important**
>
> End of support notice: On October 16, 2025, Amazon will discontinue support for CloudWatch Evidently. After October 16, 2025, you will no longer be able to access the Evidently console or Evidently resources.

A *feature* in CloudWatch Evidently represents a feature that you want to launch or that you want to test variations of.

Before you can add a feature, you must create a project. For more information, see [Create a new project](#).

To add a feature to a project

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Application Signals**, **Evidently**.
3. Choose the name of the project.
4. Choose **Add feature**.
5. For **Feature name**, enter a name to be used to identify this feature within this project.

   You can optionally add a feature description.
6. For **Feature variations**, for **Variation type** choose **Boolean**, **Long**, **Double**, or **String**. For more information, see [Variation types](#).
7. Add up to five variations for your feature. The **Value** for each variation must be valid for the **Variation type** that you selected.

   Specify one of the variations to be the default. This is the baseline that the other variations will be compared to, and should be the variation that is being served to your users now. This is also the variation that is served to users who are not added to a launch or experiment for this feature.

8. Choose **Sample code**. The code example shows what you need to add to your application to set up the variations and assign user sessions to them. You can choose between JavaScript, Java, and Python for the code.

   You don't need to add the code to your application right now, but you must do so before you start a launch or an experiment.

   For more information, see Adding code to your application.

9. (Optional) To specify that certain users always see a certain variation, choose **Overrides**, **Add override**. Then, specify a user by entering their user ID, account ID, or some other identifier in **Identifier**, and specify which variation they should see.

   This can be useful for members of your own testing team or other internal users when you want to make sure they see a specific variation. The sessions of users who are assigned overrides do not contribute to launch or experiment metrics.

   You can repeat this for as many as 20 users by choosing **Add override** again.

10. (Optional) To add tags to this feature, choose **Tags**, **Add new tag**.

    Then, for **Key**, enter a name for the tag. You can add an optional value for the tag in **Value**.

    To add another tag, choose **Add new tag** again.

    For more information, see Tagging Amazon Resources.

11. Choose **Add feature**.

Variation types

When you create a feature and define the variations, you must select a *variation type*. The possible types are:

- Boolean
- Long integer
- Double precision floating-point number
- String

The variation type sets how the different variations are differentiated in your code. You can use the variation type to simplify the implementation of CloudWatch Evidently and also to simplify the process of modifying the features in your launches and experiments.

For example, if you define a feature with the long integer variation type, the integers that you specify to differentiate the variations can be numbers passed directly into your code. One example might be testing the pixel size of a button The values for the variation types can be the number of pixels used in each variation. The code for each variation can read the variation type value and use that as the button size. To test a new button size, you can change the number used for the value of the variation, without making any other code changes.

When you set the values for your variation types within a feature, you should avoid assigning the same values to multiple variations, unless you want to do A/A testing to initially try out CloudWatch Evidently, or have other reasons to do so.

Evidently doesn't have native support for JSON as a type, but you can pass in JSON in the String variation type, and parse that JSON in your code.

Use segments to focus your audience

Important

End of support notice: On October 16, 2025, Amazon will discontinue support for CloudWatch Evidently. After October 16, 2025, you will no longer be able to access the Evidently console or Evidently resources.

You can define audience *segments* and use them in your launches and experiments. A segment is a portion of your audience that shares one or more characteristics. Examples could be Chrome browser users, users in Europe, or Firefox browser users in Europe who also fit other criteria that your application collects, such as age.

Using a segment in an experiment limits that experiment to evaluating only the users who match the segment criteria. When you use one or more segments in a launch, you can define different traffic splits for the different audience segments.

Segment rule pattern syntax

To create a segment, define a segment rule *pattern*. Specify the attributes that you want to use to evaluate whether a user session will be in the segment. The pattern that you create is compared to the value of `evaluationContext` that Evidently finds in a user session. For more information, see Using EvaluateFeature.

To create an segment rule pattern, specify the fields that you want the pattern to match. You can also use logic in your pattern, such as `And`, `Or`, `Not` and `Exists`.

For an `evaluationContext` to match a pattern, the `evaluationContext` must match all parts of the rule pattern. Evidently ignores the fields in the `evaluationContext` that aren't included in the rule pattern.

The values that rule patterns match follow JSON rules. You can include strings enclosed in quotation marks ("), numbers, and the keywords `true`, `false`, and `null`.

For strings, Evidently uses exact character-by-character matching without case-folding or any other string normalization. Therefore, rule matches are case-sensitive. For example, if your `evaluationContext` includes a `browser` attribute but your rule pattern checks for `Browser`, it will not match.

For numbers, Evidently uses string representation. For example, 300, 300.0, and 3.0e2 are not considered equal.

When you write rule patterns to match `evaluationContext`, you can use the `TestSegmentPattern` API or the `test-segment-pattern` CLI command to test that your pattern matches the correct JSON. For more information, see TestSegmentPattern.

The following summary shows all the comparison operators that are available in Evidently segment patterns.

Comparison	Example	Rule syntax
Null	UserID is null	`{ "UserID": [ null ] }`
Empty	LastName is empty	`{ "LastName": [""] }`

Comparison	Example	Rule syntax
Equals	Browser is "Chrome"	`{ "Browser": [ "Chrome" ] }`
And	Country is "France" and Device is "Mobile"	`{ "Country": [ "France" ], "Device": ["Mobile"] }`
Or (multiple values of a single attribute)	Browser is "Chrome" or "Firefox"	`{ "Browser": ["Chrome", "Firefox"] }`
Or (different attributes)	Browser is "Safari" or Device is "Tablet"	`{ "$or": [ {"Browser": ["Safari"]}, {"Device": ["Tablet"]} ] }`
Not	Browser is anything but "Safari"	`{ "Browser": [ { "anything-but": [ "Safari" ] } ] }`

Comparison	Example	Rule syntax
Numeric (equals)	Price is 100	`{ "Price": [ { "numeric": [ "=", 100 ] } ] }`
Numeric (range)	Price is more than 10, and less than or equal to 20	`{ "Price": [ { "numeric": [ ">", 10, "<=", 20 ] } ] }`
Exists	Age field exists	`{ "Age": [ { "exists": true } ] }`
Does not exist	Age field does not exist	`{ "Age": [ { "exists": false } ] }`
Begins with a prefix	Region is in the United States	`{ "Region": [ {"prefix": "us-" } ] }`
Ends with a suffix	Location has a suffix "West"	`{ "Region": [ {"suffix": "West" } ] }`

Segment rule examples

All of the following examples assume that you are passing values for `evaluationContext` with the same field labels and values that you are using in your rule patterns.

The following example matches if `Browser` is Chrome or Firefox and `Location` is US-West.

```
{
  "Browser": ["Chrome", "Firefox"],
  "Location": ["US-West"]
}
```

The following example matches if `Browser` is any browser except Chrome, the `Location` starts with `US`, and an `Age` field exists.

```
{
  "Browser": [ {"anything-but": ["Chrome"]}],
  "Location": [{"prefix": "US"}],
  "Age": [{"exists": true}]
}
```

The following example matches if the `Location` is Japan and either `Browser` is Safari or `Device` is Tablet.

```
{
 "Location": ["Japan"],
 "$or": [
   {"Browser": ["Safari"]},
   {"Device": ["Tablet"]}
 ]
}
```

Create a segment

After you create a segment, you can use it in any launch or experiment in any project.

To create a segment

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Application Signals**, **Evidently**.

3. Choose the **Segments** tab.
4. Choose **Create segment**.
5. For **Segment name**, enter a name to use to identify this segment.

   Optionally, add a description.
6. For **Segment pattern**, enter a JSON block that defines the rule pattern. For more information about rule pattern syntax, see Segment rule pattern syntax.

Create a launch

> **Important**
>
> End of support notice: On October 16, 2025, Amazon will discontinue support for CloudWatch Evidently. After October 16, 2025, you will no longer be able to access the Evidently console or Evidently resources.

To expose a new feature or change to a specified percentage of your users, create a launch. You can then monitor key metrics such as page load times and conversions before you roll out the feature to all of your users.

Before you can add a launch, you must have created a project. For more information, see Create a new project.

When you add a launch, you can use a feature that you have already created, or create a new feature while you create the launch.

To add a launch to a project

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Application Signals**, **Evidently**.
3. Select the button next to the name of the project and choose **Project actions**, **Create launch**.
4. For **Launch name**, enter a name to be used to identify this feature within this project.

   You can optionally add a description.
5. Choose either **Select from existing features** or **Add new feature**.

   If you are using an existing feature, select it under **Feature name**.

If you choose **Add new feature**, do the following:

a. For **Feature name**, enter a name to be used to identify this feature within this project. You can optionally add a description.

b. For **Feature variations**, for **Variation type** choose **Boolean**, **Long**, **Double**, or **String**. For more information, see Variation types.

c. Add up to five variations for your feature. The **Value** for each variation must be valid for the **Variation type** that you selected.

   Specify one of the variations to be the default. This is the baseline that the other variations will be compared to, and should be the variation that is being served to your users now. If you stop an experiment, this default variation will then be served to all users.

d. Choose **Sample code**. The code example shows what you need to add to your application to set up the variations and assign user sessions to them. You can choose between JavaScript, Java, and Python for the code.

   You don't need to add the code to your application right now, but you must do so before you start the launch.

   For more information, see Adding code to your application.

6. For **Launch configuration**, choose whether to start the launch immediately or schedule it to start later.

7. (Optional) To specify different traffic splits for audience segments that you have defined, instead of the traffic split that you will use for your general audience, choose **Add Segment Overrides**.

   In **Segment Overrides**, select a segment and define the traffic split to use for that segment.

   You can optionally define more segments to define traffic splits for by choosing **Add Segment Override**. A launch can have up to six segment overrides.

   For more information, see Use segments to focus your audience.

8. For **Traffic configuration**, select the traffic percentage to assign to each variation for the general audience that doesn't match the segment overrides. You can also choose to exclude variations from being served to users.

   The **Traffic summary** shows how much of your overall traffic is available for this launch.

9. If you choose to schedule the launch to start later, you can add multiple steps to the launch. Each step can use different percentages for serving the variations. To do this, choose **Add another step** and then specify the schedule and traffic percentages for the next step. You can include as many as five steps in a launch.

10. If you want to track your feature performance with metrics during the launch, choose **Metrics**, **Add metric**. You can use either CloudWatch RUM metrics or custom metrics.

    To use a custom metric, you can create the metric here using an Amazon EventBridge rule. To create a custom metric, do the following:

    - Choose **Custom metrics** and enter a name for the metric.
    - Under **Metric rule**, for **Entity ID**, enter the way to identify the entity. This can be a user or session that does an action that causes a metric value to be recorded. An example is `userDetails.userID`.
    - For **Value key**, enter the value that is to be tracked to produce the metric.
    - Optionally, enter a name for the units for the metric. This unit name is for display purposes only, for use on graphs in the Evidently console.

    As you enter those fields, the box shows examples of how to code the EventBridge rule to create the metric. For more information about EventBridge, see [What Is Amazon EventBridge?](#)

    To use RUM metrics, you must already have a RUM app monitor set up for your application. For more information, see [Set up an application to use CloudWatch RUM](#).

    > **Note**
    >
    > If you use RUM metrics, and the app monitor is not configured to sample 100% of user sessions, then not all of the user sessions that participate in the launch will send metrics to Evidently. To ensure that the launch metrics are accurate, we recommend that the app monitor uses 100% of user sessions for sampling.

11. (Optional) If you create at least one metric for the launch, you can associate an existing CloudWatch alarm with this launch. To do so, choose **Associate CloudWatch alarms**.

    When you associate an alarm with a launch, CloudWatch Evidently must add tags to the alarm with the project name and launch name. This is so that CloudWatch Evidently can display the correct alarms in the launch information in the console.

To acknowledge that CloudWatch Evidently will add these tags, choose **Allow Evidently to tag the alarm resource identified below with this launch resource.** Then, choose **Associate alarm** and enter the alarm name.

For information about creating CloudWatch alarms, see [Using Amazon CloudWatch alarms].

12. (Optional) To add tags to this launch, choose **Tags**, **Add new tag**.

    Then, for **Key**, enter a name for the tag. You can add an optional value for the tag in **Value**.

    To add another tag, choose **Add new tag** again.

    For more information, see [Tagging Amazon Resources].
13. Choose **Create launch**.

Create an experiment

> **Important**
>
> End of support notice: On October 16, 2025, Amazon will discontinue support for CloudWatch Evidently. After October 16, 2025, you will no longer be able to access the Evidently console or Evidently resources.

Use experiments to test different versions of a feature or website and collect data from real user sessions. This way, you can make choices for your application based on evidence and data.

Before you can add an experiment, you must have created a project. For more information, see [Create a new project].

When you add an experiment, you can use a feature that you have already created, or create a new feature while you create the experiment.

To add an experiment to a project

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Application Signals**, **Evidently**.
3. Select the button next to the name of the project and choose **Project actions**, **Create experiment**.

4. For **Experiment name**, enter a name to be used to identify this feature within this project.

   You can optionally add a description.

5. Choose either **Select from existing features** or **Add new feature**.

   If you are using an existing feature, select it under **Feature name**.

   If you choose **Add new feature**, do the following:

   a. For **Feature name**, enter a name to be used to identify this feature within this project. You can also optionally enter a description.
   b. For **Feature variations**, for **Variation type** choose **Boolean**, **Long**, **Double**, or **String**. The type defines which type of value is used for each variation. For more information, see Variation types.
   c. Add up to five variations for your feature. The **Value** for each variation must be valid for the **Variation type** that you selected.

      Specify one of the variations to be the default. This is the baseline that the other variations will be compared to, and should be the variation that is being served to your users now. If you stop an experiment that uses this feature, the default variation is then served to the percentage of users that were in the experiment previously.
   d. Choose **Sample code**. The code example shows what you need to add to your application to set up the variations and assign user sessions to them. You can choose between JavaScript, Java, and Python for the code.

      You don't need to add the code to your application right now, but you must do so before you start the experiment. For more information, see Adding code to your application.

6. For **Audience**, optionally select a segment that you have created if you want this experiment to apply only to the users who match that segment. For more information about segments, see Use segments to focus your audience.

7. For **Traffic split for the experiment**, specify the percentage of the selected audience whose sessions will be used in the experiment. Then allocate the traffic for the different variations that the experiment uses.

   If a launch and an experiment are both running at the same time for the same feature, the audience is first directed to the launch. Then, the percentage of traffic specified for the launch is taken from the overall audience. After that, the percentage that you specify here is the percentage of the remaining audience that is used for the experiment. Any remaining traffic after that is served the default variation.

8. For **Metrics**, choose the metrics to use to evaluate the variations during the experiment. You must use at least one metric for evaluation.

   a. For **Metric source**, choose whether to use CloudWatch RUM metrics or custom metrics.

   b. Enter a name for the metric. For **Goal**, choose **Increase** if you want a higher value for the metric to indicate a better variation. Choose **Decrease** if you want a lower value for the metric to indicate a better variation.

   c. If you are using a custom metric, you can create the metric here using an Amazon EventBridge rule. To create a custom metric, do the following:

      - Under **Metric rule**, for **Entity ID**, enter a way to identify the entity, This can be a user or session that does an action that causes a metric value to be recorded. An example is `userDetails.userID`.
      - For **Value key**, enter the value that is to be tracked to produce the metric.
      - Optionally, enter a name for the units for the metric. This unit name is for display purposes only, for use on graphs in the Evidently console.

   You can use RUM metrics only if you have set up RUM to monitor this application. For more information, see CloudWatch RUM.

   > **Note**
   >
   > If you use RUM metrics, and the app monitor is not configured to sample 100% of user sessions, then not all of the user sessions in the experiment will send metrics to Evidently. To ensure that the experiment metrics are accurate, we recommend that the app monitor uses 100% of user sessions for sampling.

   d. (Optional) To add more metrics to evaluate, choose **Add metric**. You can evaluate as many as three metrics during the experiment.

9. (Optional) To create CloudWatch alarms to use with this experiment, choose **CloudWatch alarms**. The alarms can monitor whether the difference in results between each variation and the default variation is larger than a threshold that you specify. If a variation's performance is worse than the default variation, and the difference is greater than your threshold, it goes into alarm state and notifies you.

Creating an alarm here creates one alarm for each variation that is not the default variation.

If you create an alarm, specify the following:

- For **Metric name**, choose the experiment metric to use for the alarm.
- For **Alarm condition** choose what condition causes the alarm to go into alarm state, when the variation metric values are compared to the default variation metric values. For example, choose **Greater** or **Greater/Equal** if higher numbers indicate for the variation indicates that it is performing poorly. This would be appropriate if the metric is measuring page load time, for example.
- Enter a number for the threshold, which is the percentage difference in performance that will cause the alarm to go into `ALARM` state.
- For **Average over period**, choose how much metric data for each variation is aggregated together before being compared.

You can choose **Add new alarm** again to add more alarms to the experiment.

Next, choose **Set notifications for the alarm** and select or create an Amazon Simple Notification Service topic to send alarm notifications to. For more information, see Setting up Amazon SNS notifications,

10. (Optional) To add tags to this experiment, choose **Tags**, **Add new tag**.

    Then, for **Key**, enter a name for the tag. You can add an optional value for the tag in **Value**.

    To add another tag, choose **Add new tag** again.

    For more information, see Tagging Amazon Resources.
11. Choose **Create experiment**.
12. If you haven't already, build the feature variants into your application.
13. Choose **Done**. The experiment does not start until you start it.

After you complete the steps in the following procedure, the experiment starts immediately.

To start an experiment that you have created

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.

2. In the navigation pane, choose **Application Signals**, **Evidently**.
3. Choose the name of the project.
4. Choose the **Experiments** tab.
5. Choose the button next to the name of the experiment, and choose **Actions**, **Start experiment**.
6. (Optional) To view or modify the experiment settings you made when you created it, choose **Experiment setup**.
7. Choose a time for the experiment to end.
8. Choose **Start experiment**.

   The experiment starts immediately.

Manage features, launches, and experiments

> **Important**
>
> End of support notice: On October 16, 2025, Amazon will discontinue support for CloudWatch Evidently. After October 16, 2025, you will no longer be able to access the Evidently console or Evidently resources.

Use the procedures in these sections to manage the features, launches, and experiments that you have created.

Topics

- See the current evaluation rules and audience traffic for a feature
- Modify launch traffic
- Modify a launch's future steps
- Modify experiment traffic
- Stop a launch
- Stop an experiment

See the current evaluation rules and audience traffic for a feature

> **Important**
>
> End of support notice: On October 16, 2025, Amazon will discontinue support for CloudWatch Evidently. After October 16, 2025, you will no longer be able to access the Evidently console or Evidently resources.

You can use the CloudWatch Evidently console to see how the feature's evaluation rules are allocating the audience traffic among the feature's current launches, experiments, and variations.

To view the audience traffic for a feature

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Application Signals**, **Evidently**.
3. Choose the name of the project that contains the feature.
4. Choose the **Features** tab.
5. Choose the name of the feature.

   In the **Evaluation rules** tab, you can see the flow of audience traffic for your feature, as follows:

   - First, the overrides are evaluated. These specify that certain users are always served a specific variation. The sessions of users who are assigned overrides do not contribute to launch or experiment metrics.
   - Next, the remaining traffic is available for the ongoing launch, if there is one. If there is a launch in progress, the table in the **Launches** section displays the launch name and the launch traffic split among the feature variations. On the right side of the **Launches** section, a **Traffic** indicator displays how much of the available audience (after overrides) is allocated to this launch. The rest of the traffic not allocated to the launch flows to the experiment (if any) and then the default variation.
   - Next, the remaining traffic is available for the ongoing experiment, if there is one. If there is an experiment in progress, the table in the **Experiments** section displays the experiment name and progress. On the right side of the **Experiments** section, a **Traffic** indicator displays how much of the available audience (after overrides and launches) is allocated to this

experiment. The rest of the traffic not allocated to the launch or the experiment is served the default variation of the feature.

Modify launch traffic

> **Important**
>
> End of support notice: On October 16, 2025, Amazon will discontinue support for CloudWatch Evidently. After October 16, 2025, you will no longer be able to access the Evidently console or Evidently resources.

You can modify the traffic allocation for a launch at any time, including while the launch is ongoing.

If you have both an ongoing launch and an ongoing experiment for the same feature, any changes to the feature traffic will cause a change in the experiment traffic. This is because the audience available to the experiment is the portion of your total audience that is not already allocated to the launch. Increasing launch traffic will descrease the audience available to the experiment, and decreasing launch traffic or ending the launch will increase the audience available to the experiment.

To modify the traffic allocation for a launch

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Application Signals**, **Evidently**.
3. Choose the name of the project that contains the launch.
4. Choose the **Launches** tab.
5. Choose the name of the launch.

   Choose **Modify launch traffic**.
6. For **Serve**, select the new traffic percentage to assign to each variation. You can also choose to exclude variations from being served to users. As you change these values, you can see the updated effects on your overall feature traffic under **Traffic summary**.

   The **Traffic summary** shows how much of your overall traffic is available for this launch, and how much of that available traffic is allocated to this launch.

7. Choose **Modify**.

Modify a launch's future steps

> **Important**
>
> End of support notice: On October 16, 2025, Amazon will discontinue support for CloudWatch Evidently. After October 16, 2025, you will no longer be able to access the Evidently console or Evidently resources.

You can modify the configuration of launch steps that haven't happened yet, and also add more steps to a launch.

To modify the steps for a launch

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Application Signals**, **Evidently**.
3. Choose the name of the project that contains the launch.
4. Choose the **Launches** tab.
5. Choose the name of the launch.

   Choose **Modify launch traffic**.
6. Choose **Schedule launch**.
7. For any steps that have not started yet, you can modify the percentage of the available audience to use in the experiment. You can also modify how their traffic is allocated among the variations.

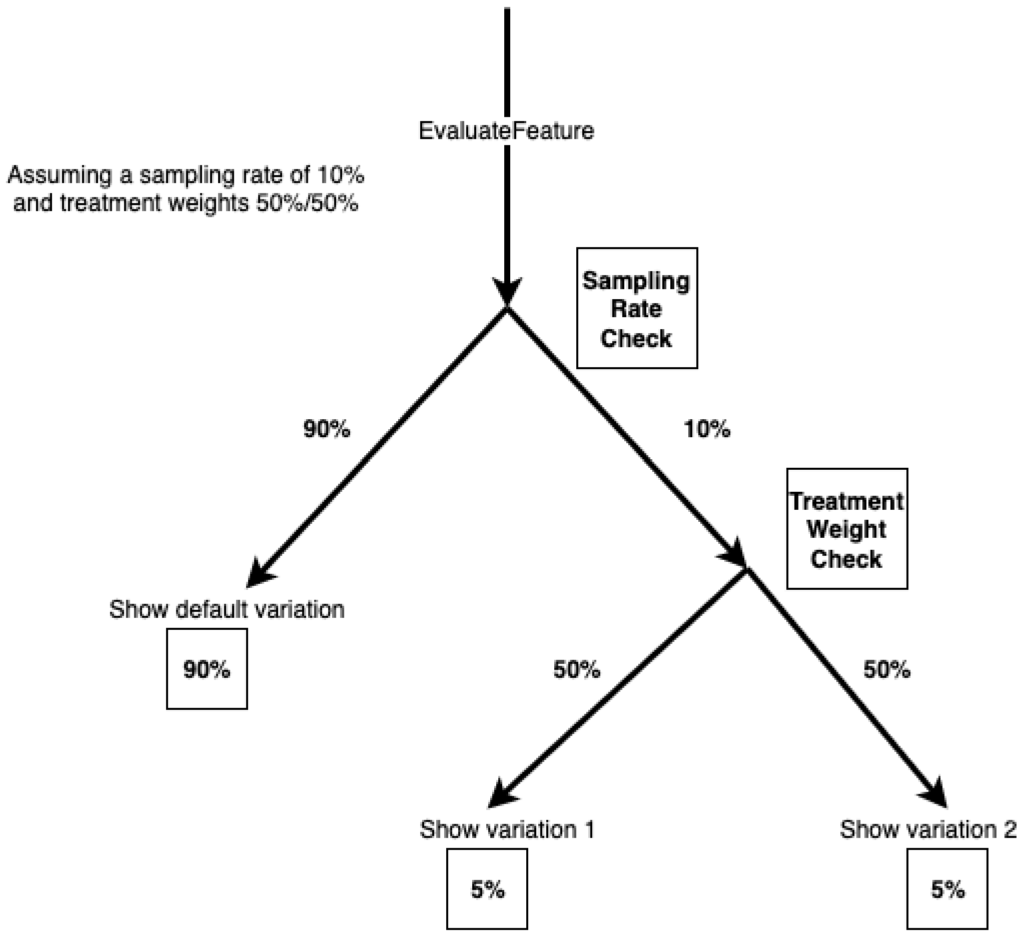
   You can add more steps to the launch by choosing **Add another step**. A launch can have a maximum of five steps.
8. Choose **Modify**.

Modify experiment traffic

> **Important**
>
> End of support notice: On October 16, 2025, Amazon will discontinue support for CloudWatch Evidently. After October 16, 2025, you will no longer be able to access the Evidently console or Evidently resources.

You can modify the sampling rate for an experiment at any time, including while the experiment is ongoing. However, you can't update the treatment weights after an experiment is running. Therefore, you can change the total traffic exposed to the experiment after an experiment is running, but not the relative allocation to each treatment. If you modify the traffic of an ongoing experiment, we recommend that you only increase the traffic allocation, so that you don't introduce bias.

The following diagram shows how client traffic is allocated to different variations in an experiment. In this experiment, the sampling rate is 10% and the treatment weights for the two variations are 50% each.



To modify the traffic allocation for an experiment

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Application monitoring**, **Evidently**.
3. Choose the name of the project that contains the launch.
4. Choose the **Experiments** tab.
5. Choose the name of the launch.
6. Choose **Modify experiment traffic**.
7. Enter a percentage or use the slider to specify how much of the available traffic to allocate to this experiment. The available traffic is the total audience minus the traffic that is allocated to a current launch, if there is one. The traffic that is not allocated to the launch or experiment is served the default variation.
8. Choose **Modify**.

Stop a launch

> **Important**
>
> End of support notice: On October 16, 2025, Amazon will discontinue support for CloudWatch Evidently. After October 16, 2025, you will no longer be able to access the Evidently console or Evidently resources.

If you stop an ongoing launch, you will not be able to resume it or restart it. Also, it will not be evaluated as a rule for traffic allocation, and the traffic that was allocated to the launch will instead be available to the feature's experiment, if there is one. Otherwise, all traffic will be served the default variation after the launch is stopped.

To permanently stop a launch

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Application Signals**, **Evidently**.
3. Choose the name of the project that contains the launch.
4. Choose the **Launch** tab.
5. Choose the button to the left of the name of the launch.
6. Choose **Actions**, **Cancel launch** or **Actions**, **Mark as complete**.

Stop an experiment

> **Important**
>
> End of support notice: On October 16, 2025, Amazon will discontinue support for CloudWatch Evidently. After October 16, 2025, you will no longer be able to access the Evidently console or Evidently resources.

If you stop an ongoing experiment, you will not be able to resume it or restart it. The portion of traffic that was previously used in the experiment will be served the default variation.

When an experiment is not manually stopped and passes its end date, the traffic does not change. The portion of traffic allocated to the experiment still goes to the experiment. To stop this, and cause the experiment traffic to instead be served the default variation, mark the experiment as complete.

When you stop an experiment, you can choose to cancel it or mark it as complete. If you cancel, it will be shown as **Cancelled** in the list of experiments. If you choose to mark it as complete, it is shown as **Completed**.

To permanently stop an experiment

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Application Signals**, **Evidently**.
3. Choose the name of the project that contains the experiment.
4. Choose the **Experiments** tab.
5. Choose the button to the left of the name of the experiment.
6. Choose **Actions**, **Cancel experiment** or **Actions**, **Mark as complete**.

Adding code to your application

> **Important**
>
> End of support notice: On October 16, 2025, Amazon will discontinue support for CloudWatch Evidently. After October 16, 2025, you will no longer be able to access the Evidently console or Evidently resources.

To work with CloudWatch Evidently, you add code to your application to assign a variation to each user session, and to send metrics to Evidently. You use the CloudWatch Evidently `EvaluateFeature` operation to assign variations to user sessions, and you use the `PutProjectEvents` operation to send events to Evidently to be used to calculate metrics for your launches or experiments.

When you create variations or custom metrics, the CloudWatch Evidently console provides samples of the code you need to add.

For an end-to-end example, see Tutorial: A/B testing with the Evidently sample application.

Using EvaluateFeature

When feature variations are used in a launch or experiment, the application uses the EvaluateFeature operation to assign each user session a variation. The assignment of a variation to a user is an *evaluation event*. When you call this operation, you pass in the following:

- **Feature name**– Required. Evidently processes the evaluation according to the feature evaluation rules of the launch or experiment, and selects a variation for the entity.
- **entityId**– Required. Represents a unique user.
- **evaluationContext**– Optional. A JSON object representing additional information about a user. Evidently will use this value to match the user to a *segment* of your audience during feature evaluations, if you have created segments. For more information, see Use segments to focus your audience.

  The following is an example of an `evaluationContext` value that you can send to Evidently.

```
{
    "Browser": "Chrome",
    "Location": {
        "Country": "United States",
        "Zipcode": 98007
    }
}
```

Sticky evaluations

CloudWatch Evidently uses "sticky" evaluations. A single configuration of `entityId`, feature, *feature configuration*, and `evaluationContext` always receives the same variation assignment. The only time this variation assignment changes is when an entity is added to an override or the experiment traffic is dialed up.

A feature configuration includes the following:

- The feature variations
- The variation configuration (percentages assigned to each variation) for a currently running experiment for this feature, if any.
- The variation configuration for a currently running launch for this feature, if any. The variation configuration includes the defined segment overrides, if any.

If an experiment's traffic allocation is *increased*, any `entityId` that was previously assigned to an experiment treatment group will continue to receive the same treatment. Any `entityId` that was previously assigned to the control group, might be assigned to an experiment treatment group, according to the variation configuration specified for the experiment.

If an experiment's traffic allocation is *decreased*, an `entityId` might go from a treatment group to a control group, but it will not go into a different treatment group.

Using PutProjectEvents

To code a custom metric for Evidently, you use the [PutProjectEvents] operation. The following is a simple payload example.

```
{
    "events": [
        {
            "timestamp": {{$timestamp}},
            "type": "aws.evidently.custom",
            "data": "{\"details\": {\"pageLoadTime\": 800.0}, \"userDetails\":
 {\"userId\": \"test-user\"}}"
        }
    ]
}
```

The `entityIdKey` can just be an `entityId` or you can rename it to anything else, such as `userId`. In the actual event, `entityId` can be a username, a session ID, and so on.

```
"metricDefinition":{
    "name": "noFilter",
    "entityIdKey": "userDetails.userId", //should be consistent with jsonValue in
 events "data" fields
    "valueKey": "details.pageLoadTime"
},
```

To ensure that events are associated with the correct launch or experiment, you must pass the same `entityId` when you call both `EvaluateFeature` and `PutProjectEvents`. Be sure to call `PutProjectEvents` after the `EvaluateFeature` call, otherwise data is dropped and won't be used by CloudWatch Evidently.

The `PutProjectEvents` operation does not require the feature name as an input parameter. This way, you can use a single event in multiple experiments. For example, suppose you call

`EvaluateFeature` with the `entityId` set to `userDetails.userId`. If you have two or more experiments running, you can have a single event from that user's session emit metrics for each of those experiments. To do this, you call `PutProjectEvents` once for each experiment, using that same `entityId`.

Timing

After your application calls `EvaluateFeature`, there is a one-hour time period where metric events from `PutProjectEvents` are attributed based on that evaluation. If any more events occur after the one-hour period, they are not attributed.

However, if the same `entityId` is used for a new `EvaluateFeature` call during that initial call's one-hour window, the later `EvaluateFeature` result is now used instead, and the one-hour timer is restarted. This can only happen in certain circumstances, such as when experiment traffic is dialed up between the two assignments, as explained in the previous **Sticky evaluations** section.

For an end-to-end example, see [Tutorial: A/B testing with the Evidently sample application](#).

Evidently project data storage in CloudWatch

> **Important**
>
> End of support notice: On October 16, 2025, Amazon will discontinue support for CloudWatch Evidently. After October 16, 2025, you will no longer be able to access the Evidently console or Evidently resources.

Evidently collects two types of events:

- **Evaluation events** are related to which feature variation is assigned to a user session. Evidently uses these events to produce metrics and other experiment and launch data, which you can view in the Evidently console.

  You can also choose to store these evaluation events in Amazon CloudWatch Logs or Amazon S3.
- **Custom events** are used to generate metrics from user actions such as clicks and checkouts. Evidently doesn't provide a method for you to store custom events. If you want to save them, you must modify your application code to send them to a storage option outside of Evidently.

Format of evaluation event logs

If you choose to store evaluation events in CloudWatch Logs or Amazon S3, each evaluation event is stored as a log event with the following format:

```
{
    "event_timestamp": 1642624900215,
    "event_type": "evaluation",
    "version": "1.0.0",
    "project_arn": "arn:aws:evidently:us-east-1:123456789012:project/petfood",
    "feature": "petfood-upsell-text",
    "variation": "Variation1",
    "entity_id": "7",
    "entity_attributes": {},
    "evaluation_type": "EXPERIMENT_RULE_MATCH",
    "treatment": "Variation1",
    "experiment": "petfood-experiment-2"
}
```

Here are more details about the preceding evaluation event format:

- The timestamp is in UNIX time with milliseconds
- The variation is the name of the variation of the feature which was assigned to this user session.
- The entity ID is a string.
- Entity attributes are a hash of arbitrary values sent by the client. For example, if the `entityId` is mapped to blue or green, then you can optionally send userIDs, session data, or whatever else that you want want from a correlation and data warehouse perspective.

IAM policy and encryption for evaluation event storage in Amazon S3

If you choose to use Amazon S3 to store evaluation events, you must add an IAM policy like the following to allow Evidently to publish logs to the Amazon S3 bucket. This is because Amazon S3 buckets and the objects they contain are private, and they don't allow access to other services by default.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AWSLogDeliveryWrite",
```

```
            "Effect": "Allow",
            "Principal": {"Service": "delivery.logs.amazonaws.com"},
            "Action": "s3:PutObject",
            "Resource": "arn:aws:s3:::bucket_name/optional_folder/AWSLogs/account_id/
*",
            "Condition": {"StringEquals": {"s3:x-amz-acl": "bucket-owner-full-
control"}}
        },
        {
            "Sid": "AWSLogDeliveryCheck",
            "Effect": "Allow",
            "Principal": {"Service": "delivery.logs.amazonaws.com"},
            "Action": ["s3:GetBucketAcl", "s3:ListBucket"],
            "Resource": "arn:aws:s3:::bucket_name"
        }
    ]
}
```

If you store Evidently data in Amazon S3, you can also choose to encrypt it with Server-Side Encryption with Amazon Key Management Service Keys (SSE-KMS). For more information, see Protecting data using server-side encryption.

If you use a customer managed key from Amazon KMS, you must add the following to the IAM policy for your key. This allows Evidently to write to the bucket.

```
{
    "Sid": "AllowEvidentlyToUseCustomerManagedKey",
    "Effect": "Allow",
    "Principal": {
        "Service": [
            "delivery.logs.amazonaws.com"
        ]
    },
   "Action": [
       "kms:Encrypt",
       "kms:Decrypt",
       "kms:ReEncrypt*",
       "kms:GenerateDataKey*",
       "kms:DescribeKey"
    ],
    "Resource": "*"
}
```

How Evidently calculates results

> **Important**
>
> End of support notice: On October 16, 2025, Amazon will discontinue support for CloudWatch Evidently. After October 16, 2025, you will no longer be able to access the Evidently console or Evidently resources.

You can use Amazon CloudWatch Evidently A/B testing as a tool for data-driven decision making. In an A/B test, users are randomly assigned to either the control group (also called the default variation), or one of the treatment groups (also called the tested variations). For example, users in the control group might experience the website, service, or application in the same way that they did before the experiment started. Meanwhile, users in the treatment group might experience the change.

CloudWatch Evidently supports up to five different variations in an experiment. Evidently randomly assigns traffic to these variations. This way, you can track business metrics (such as revenue) and performance metrics (such as latency) for each group. Evidently does the following:

- Compares the treatment with the control. (For example, compares whether revenue increases or decreases with a new checkout process.)
- Indicates whether the observed difference between the treatment and the control is *significant*. For this, Evidently offers two approaches: *Frequentist significance levels* and *Bayesian probabilities*.

Why use Frequentist and Bayesian approaches?

Consider a case where the treatment has no effect compared to the control, or a case where the treatment is identical to the control (an *A/A test*). You would still observe a small difference between the treatment and the control in the data. This is because the test participants consist of a finite sample of users, representing a small percentage of all users of the website, service, or application. Frequentist significance levels and Bayesian probabilities provide insights into whether the observed difference is significant or due to chance.

Evidently considers the following to determine whether the observed difference is significant:

- How big the difference is

- How many samples are part of the test
- How the data is distributed

Frequentist analysis in Evidently

Evidently uses sequential testing, which avoids the usual problems of *peeking*, a common pitfall of frequentist statistics. Peeking is the practice of checking the results of an ongoing A/B test in order to stop it and make a decision based on the observed results. For more information about sequential testing, see Time-uniform, nonparametric, nonasymptotic confidence sequences by Howard et al. (Ann. Statist. 49 (2) 1055 - 1080, 2021).

Because Evidently's results are valid at any time (*anytime-valid* results), you can peek at results during the experiment and still draw sound conclusions. This can reduce some of the costs of experimentation, because you can stop an experiment before the scheduled time if the results already have significance.

Evidently generates anytime-valid significance levels and anytime-valid 95% confidence intervals of the difference between the tested variation and the default variation in the target metric. The **Result** column in the experiment results indicates the tested variation performance, which can be one of the following:

- **Inconclusive** – The significance level is less than 95%
- **Better** – The significance level is 95% or higher and one of the following is true:
  - The lower bound of the 95%-confidence interval is higher than zero and the metric should increase
  - The upper bound of the 95%-confidence interval is lower than zero and the metric should decrease
- **Worse** – The significance level is 95% or higher and one of the following is true:
  - The upper bound of the 95%-confidence interval is higher than zero and the metric should increase
  - The lower bound of the 95%-confidence interval is lower than zero and the metric should decrease
- **Best** – The experiment has two or more tested variations in addition to the default variation, and the following conditions are met:
  - The variation qualifies for the *Better* designation
  - One of the following is true:

- The lower bound of the 95%-confidence interval is higher than the upper bound of the 95%-confidence intervals of all the other variations and the metric should increase
- The upper bound of the 95%-confidence interval is lower than the lower bound of the 95%-confidence intervals of all the other variations and the metric should decrease

Bayesian analysis in Evidently

With Bayesian analysis, you can calculate the probability that the mean in the tested variation is larger or smaller than the mean in the default variation. Evidently performs Bayesian inference for the mean of the target metric by using *conjugate priors*. With conjugate priors, Evidently can more efficiently infer the posterior distribution needed for the Bayesian analysis.

Evidently waits until the end date of the experiment to compute the results of the Bayesian analysis. The results page displays the following:

- *probability of increase* – The probability that the mean of the metric in the tested variation is at least 3% larger than the mean in the default variation
- *probability of decrease* – The probability that the mean of the metric in the tested variation is at least 3% smaller than the mean in the default variation
- *probability of no change* – The probability that the mean of the metric in the tested variation lies within ±3% of the mean in the default variation

The **Result** column indicates the performance of the variation, and can be one of the following:

- **Better** – The probability of increase is at least 90% and the metric should increase, or the probability of decrease is at least 90% and the metric should decrease
- **Worse** – The probability of decrease is at least 90% and the metric should increase, or the probability of increase is at least 90% and the metric should decrease

View launch results in the dashboard

> **Important**
>
> End of support notice: On October 16, 2025, Amazon will discontinue support for CloudWatch Evidently. After October 16, 2025, you will no longer be able to access the Evidently console or Evidently resources.

You can see the progress and metric results of an experiment while it is ongoing and after it is completed.

To see the progress and results of a launch

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Application Signals**, **Evidently**.
3. Choose the name of the project that contains the launch.
4. Choose the **Launch** tab.
5. Choose the name of the launch.
6. To see the launch steps and the traffic allocations for each step, choose the **Launch** tab.
7. To see the number of user sessions assigned to each variation over time, and to view the performance metrics for each variation in the launch, choose the **Monitoring** tab.

   This view also displays whether any launch alarms have gone into `ALARM` state during the launch.
8. To see the variations, metrics, alarms, and tags for this launch, choose the **Configuration** tab.

View experiment results in the dashboard

> **Important**
>
> End of support notice: On October 16, 2025, Amazon will discontinue support for CloudWatch Evidently. After October 16, 2025, you will no longer be able to access the Evidently console or Evidently resources.

You can see the statistical results of an experiment while it is ongoing and after it is completed. Experiment results are available up to 63 days after the start of the experiment. They are not available after that because of CloudWatch data retention policies.

No statistical results are displayed until each variation has at least 100 events.

Evidently performs an additional offline p-value analysis at the end of the experiment. Offline p-value analysis can detect statistical significance in some cases where the anytime p-values used during the experiment do not find statistical significance.

For more information about how CloudWatch Evidently calculates experiment results, see How Evidently calculates results.

To see the results of an experiment

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Application Signals**, **Evidently**.
3. Choose the name of the project that contains the experiment.
4. Choose the **Experiments** tab.
5. Choose the name of the experiment, and then choose the **Results** tab.
6. By **Variation performance**, there is a control where you can select which experiment statistics to display. If you select more than one statistic, Evidently displays a graph and table for each statistic.

   Each graph and table displays the results of the experiment so far.

   Each graph can display the following results. You can use the control at the right of the graph to determine which of the following items is displayed:

   - The number of user session events recorded for each variation.
   - The average value of the metric that is selected at the top of the graph, for each variation.
   - The statistical significance of the experiments. This compares the difference for the metric selected at the top of the graph with the default variation and each of the other variations.
   - The 95% upper and lower confidence bounds on the difference of the selected metric, between each of the variations and the default variation.

   The table displays a row for each variation. For each variation that is not the default, Evidently displays whether it has received enough data to declare the results statistically significant. It also shows whether the variation's improvement in the statistical value has reached a 95% confidence level.

Finally, in the **Result** column, Evidently provides a recommendation about which variation performs best based on this statistic, or whether the results are inconclusive.

How CloudWatch Evidently collects and stores data

> **Important**
>
> End of support notice: On October 16, 2025, Amazon will discontinue support for CloudWatch Evidently. After October 16, 2025, you will no longer be able to access the Evidently console or Evidently resources.

Amazon CloudWatch Evidently collects and stores data related to project configurations so that customers can run experiments and launches. The data includes the following:

- Metadata about projects, features, launches, and experiments
- Metric events
- Evaluation data

Resource metadata is stored in Amazon DynamoDB. The data is encrypted at rest by default, using Amazon owned keys. These keys are a collection of Amazon KMS keys that an Amazon Web Services service owns and manages for use in multiple Amazon Web Services accounts. Customers can’t view, manage, or audit the use of these keys. Customers are also not required to take action or change programs to protect the keys that encrypt their data.

For more information, see Amazon owned keys in the Amazon Key Management Service Developer Guide.

Evidently metric events and evaluation events are delivered directly to customer-owned locations.

Data in transit is automatically encrypted with HTTPS. This data will be delivered to customer-owned locations.

You can also choose to store evaluation events in Amazon Simple Storage Service or Amazon CloudWatch Logs. For more information about how you can secure your data in these services, see

Enabling Amazon S3 default bucket encryption and Encrypting log data in CloudWatch Logs using Amazon KMS.

Retrieving data

You can retrieve your data using CloudWatch Evidently APIs. To retrieve project data, use GetProject or ListProjects.

To retrieve feature data, use GetFeature or ListFeatures.

To retrieve launch data, use GetLaunch or ListLaunches.

To retrieve experiment data, use GetExperiment, ListExperiments, or GetExperimentResults.

Modifying and deleting data

You can modify and delete your data using CloudWatch Evidently APIs. For project data, use UpdateProject or DeleteProject.

For feature data, use UpdateFeature or DeleteFeature.

For launch data, use UpdateLaunch or DeleteLaunch.

For experiment data, use UpdateExperiment or DeleteExperiment.

Using service-linked roles for Evidently

> **Important**
>
> End of support notice: On October 16, 2025, Amazon will discontinue support for CloudWatch Evidently. After October 16, 2025, you will no longer be able to access the Evidently console or Evidently resources.

CloudWatch Evidently uses Amazon Identity and Access Management (IAM) service-linked roles. A service-linked role is a unique type of IAM role that is linked directly to Evidently. Service-linked roles are predefined by Evidently and include all the permissions that the service requires to call other Amazon services on your behalf.

A service-linked role makes setting up Evidently easier because you don't have to manually add the necessary permissions. Evidently defines the permissions of its service-linked roles, and unless defined otherwise, only Evidently can assume its roles. The defined permissions include the trust policy and the permissions policy, and that permissions policy cannot be attached to any other IAM entity.

You can delete a service-linked role only after first deleting its related resources. This protects your Evidently resources because you can't inadvertently remove permission to access the resources.

For information about other services that support service-linked roles, see Amazon Services That Work with IAM and look for the services that have **Yes** in the **Service-linked roles** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

Service-linked role permissions for Evidently

Evidently uses the service-linked role named **AWSServiceRoleForCloudWatchEvidently –** Allows CloudWatch Evidently to manage associated Amazon resourcees on behalf of the customer.

The AWSServiceRoleForCloudWatchEvidently service-linked role trusts the following services to assume the role:

- `CloudWatch Evidently`

The role permissions policy named AmazonCloudWatchEvidentlyServiceRolePolicy allows Evidently to complete the following actions on the specified resources:

- Actions: `appconfig:StartDeployment`, `appconfig:StopDeployment`, `appconfig:ListDeployments`, and `appconfig:TagResource` on Evidently thick clients.

You must configure permissions to allow an IAM entity (such as a user, group, or role) to create, edit, or delete a service-linked role. For more information, see Service-linked role permissions in the *IAM User Guide*.

Creating a service-linked role for Evidently

You don't need to manually create a service-linked role. When you start using an Evidently thick client in the Amazon Web Services Management Console, the Amazon CLI, or the Amazon API, Evidently creates the service-linked role for you.

If you delete this service-linked role, and then need to create it again, you can use the same process to recreate the role in your account. When you start using an Evidently thick client, Evidently creates the service-linked role for you again.

Editing a service-linked role for Evidently

Evidently does not allow you to edit the AWSServiceRoleForCloudWatchEvidently service-linked role. After you create a service-linked role, you cannot change the name of the role because various entities might reference the role. However, you can edit the description of the role using IAM. For more information, see [Editing a service-linked role](#) in the *IAM User Guide*.

Deleting a service-linked role for Evidently

If you no longer need to use a feature or service that requires a service-linked role, we recommend that you delete that role. That way you don't have an unused entity that is not actively monitored or maintained. However, you must clean up the resources for your service-linked role before you can manually delete it. You must delete all Evidently projects that are using thick clients.

> **Note**
>
> If the Evidently service is using the role when you try to delete the resources, then the deletion might fail. If that happens, wait for a few minutes and try the operation again.

To delete Evidently resources used by AWSServiceRoleForCloudWatchEvidently

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Application monitoring**, **Evidently**.
3. In the list of projects, select the check box next to the projects that used thick clients.
4. Choose **Project actions**, **Delete project**.

To manually delete the service-linked role using IAM

Use the IAM console, the Amazon CLI, or the Amazon API to delete the AWSServiceRoleForCloudWatchEvidently service-linked role. For more information, see Deleting a service-linked role in the *IAM User Guide*.

Supported regions for Evidently service-linked roles

Evidently supports using service-linked roles in all of the regions where the service is available. For more information, see Amazon regions and endpoints.

Evidently quotas in CloudWatch

> **Important**
>
> End of support notice: On October 16, 2025, Amazon will discontinue support for CloudWatch Evidently. After October 16, 2025, you will no longer be able to access the Evidently console or Evidently resources.

CloudWatch Evidently has the following quotas.

Resource	Default quota
Projects	50 per Region per account You can request a quota increase.
Segments	500 per Region per account You can request a quota increase.
Quotas per project	• 100 total features • 500 total launches • 50 running launches • 500 total experiments • 50 running experiments You can request a quota increase for all of these quotas.
API quotas (all quotas are per Region)	• PutProjectEvents: 1000 transactions per second (TPS) in US East (N. Virginia), US West (Oregon), and Europe (Ireland). 200 TPS in all other Regions. • EvaluateFeature: 1000 TPS in US East (N. Virginia), US West (Oregon), and Europe (Ireland). 200 TPS in all other Regions. • BatchEvaluateFeature: 50 TPS

Resource	Default quota
	• Create, Read, Update, Delete (CRUD) APIs: 10 TPS combined across all CRUD APIs You can request a quota increase for all of these quotas.

Tutorial: A/B testing with the Evidently sample application

Important

End of support notice: On October 16, 2025, Amazon will discontinue support for CloudWatch Evidently. After October 16, 2025, you will no longer be able to access the Evidently console or Evidently resources.

This section provides a tutorial for using Amazon CloudWatch Evidently for A/B testing. This tutorial the Evidently sample application, which is a simple react application. The sample app will be configured to either display a `showDiscount` feature or not. When the feature is shown to a user, the price displayed on the shopping website it shown at a 20% discount.

In addition to showing the discount to some users and not to others, in this tutorial you set up Evidently to collect page load time metrics from both variations.

Warning

This scenario requires IAM users with programmatic access and long-term credentials, which presents a security risk. To help mitigate this risk, we recommend that you provide these users with only the permissions they require to perform the task and that you remove these users when they are no longer needed. Access keys can be updated if necessary. For more information, see Update access keys in the *IAM User Guide*.

Step 1: Download the sample application

Start by downloading the Evidently sample application.

To download the sample application

1. Download the sample application from the following Amazon S3 bucket:

```
https://evidently-sample-application.s3.us-west-2.amazonaws.com/evidently-sample-
shopping-app.zip
```

2. Unzip the package.

Step 2: Add the Evidently endpoint and set up credentials

Next, add the Region and endpoint for Evidently to the `config.js` file in the `src` directory in the sample app package, as in the following example:

```
evidently: {
    REGION: "us-west-2",
    ENDPOINT: "https://evidently.us-west-2.amazonaws.com (https://evidently.us-
west-2.amazonaws.com/)",
},
```

You also must make sure that the application has permission to call CloudWatch Evidently.

To grant the sample app permissions to call Evidently

1. Federate to your Amazon account.
2. Create an IAM user and attach the **AmazonCloudWatchEvidentlyFullAccess** policy to this user.
3. Make a note of the IAM user's access key id and secret access key, because you will need them in the next step.
4. In the same `config.js` file that you modified earlier in this section, enter the values of the access key ID and the secrect access key, as in the following example:

```
credential: {
   accessKeyId: "Access key ID",
   secretAccessKey: "Secret key"
}
```

> **Important**
>
> We use this step to make the sample app as simple as possible for you to try out. We do not recommend that you put your IAM user credential into your actual production application. Instead, we recommend that you use Amazon Cognito for authentication. For more information, see [Integrating Amazon Cognito with web and mobile apps](#).

Step 3: Set up code for the feature evaluation

When you use CloudWatch Evidently to evaluate a feature, you must use the **EvaluateFeature** operation to randomly select a feature variation for each user session. This operation assigns user sessions to each variation of the feature, according to the percentages that you specified in the experiment.

To set up the feature evaluation code for the bookstore demo app

1. Add the client builder in the `src/App.jsx` file so that the sample app can call Evidently.

```
import Evidently from 'aws-sdk/clients/evidently';
import config from './config';

const defaultClientBuilder = (
    endpoint,
    region,
) => {
    const credentials = {
        accessKeyId: config.credential.accessKeyId,
        secretAccessKey: config.credential.secretAccessKey
    }
    return new Evidently({
        endpoint,
        region,
        credentials,
    });
};
```

2. Add the following to the `const App` code section to initiate the client.

```
  if (client == null) {
```

```
      client = defaultClientBuilder(
        config.evidently.ENDPOINT,
        config.evidently.REGION,
      );
```

3. Construct `evaluateFeatureRequest` by adding the following code. This code pre-fills the project name and feature name that we recommend later in this tutorial. You can substitute your own project and feature names, as long as you also specify those project and feature names in the Evidently console.

```
const evaluateFeatureRequest = {
              entityId: id,
              // Input Your feature name
              feature: 'showDiscount',
              // Input Your project name'
              project: 'EvidentlySampleApp',
            };
```

4. Add the code to call Evidently for feature evaluation. When the request is sent, Evidently randomly assigns the user session to either see the `showDiscount` feature or not.

```
client.evaluateFeature(evaluateFeatureRequest).promise().then(res => {
    if(res.value?.boolValue !== undefined) {
        setShowDiscount(res.value.boolValue);
    }
    getPageLoadTime()
 })
```

Step 4: Set up code for the experiment metrics

For the custom metric, use Evidently's `PutProjectEvents` API to send metric results to Evidently. The following examples show how to set up the custom metric and send experiment data to Evidently.

Add the following function to calculate the page load time and use `PutProjectEvents` to send the metric values to Evidently. Add the following function to into `Home.tsx` and call this function within the `EvaluateFeature` API:

```
const getPageLoadTime = () => {
    const timeSpent = (new Date().getTime() - startTime.getTime()) * 1.000001;
```

```
    const pageLoadTimeData = `{
      "details": {
        "pageLoadTime": ${timeSpent}
      },
      "UserDetails": { "userId": "${id}", "sessionId": "${id}"}
    }`;
    const putProjectEventsRequest = {
      project: 'EvidentlySampleApp',
      events: [
        {
          timestamp: new Date(),
          type: 'aws.evidently.custom',
          data: JSON.parse(pageLoadTimeData)
        },
      ],
    };
    client.putProjectEvents(putProjectEventsRequest).promise();
  }
```

Here is what the `App.js` file should look like after all the editing that you have done since downloading it.

```
import React, { useEffect, useState } from "react";
import { BrowserRouter as Router, Switch } from "react-router-dom";
import AuthProvider from "contexts/auth";
import CommonProvider from "contexts/common";
import ProductsProvider from "contexts/products";
import CartProvider from "contexts/cart";
import CheckoutProvider from "contexts/checkout";
import RouteWrapper from "layouts/RouteWrapper";
import AuthLayout from "layouts/AuthLayout";
import CommonLayout from "layouts/CommonLayout";
import AuthPage from "pages/auth";
import HomePage from "pages/home";
import CheckoutPage from "pages/checkout";
import "assets/scss/style.scss";
import { Spinner } from 'react-bootstrap';

import Evidently from 'aws-sdk/clients/evidently';
import config from './config';

const defaultClientBuilder = (
  endpoint,
```

```
  region,
) => {
  const credentials = {
    accessKeyId: config.credential.accessKeyId,
    secretAccessKey: config.credential.secretAccessKey
  }
  return new Evidently({
    endpoint,
    region,
    credentials,
  });
};

const App = () => {
  const [isLoading, setIsLoading] = useState(true);
  const [startTime, setStartTime] = useState(new Date());
  const [showDiscount, setShowDiscount] = useState(false);
  let client = null;
  let id = null;

  useEffect(() => {
    id = new Date().getTime().toString();
    setStartTime(new Date());
    if (client == null) {
      client = defaultClientBuilder(
        config.evidently.ENDPOINT,
        config.evidently.REGION,
      );
    }
    const evaluateFeatureRequest = {
      entityId: id,
      // Input Your feature name
      feature: 'showDiscount',
      // Input Your project name'
      project: 'EvidentlySampleApp',
    };

    // Launch
    client.evaluateFeature(evaluateFeatureRequest).promise().then(res => {
      if(res.value?.boolValue !== undefined) {
        setShowDiscount(res.value.boolValue);
      }
    });
```

```
    // Experiment
    client.evaluateFeature(evaluateFeatureRequest).promise().then(res => {
      if(res.value?.boolValue !== undefined) {
        setShowDiscount(res.value.boolValue);
      }
      getPageLoadTime()
    })

    setIsLoading(false);
  },[]);

  const getPageLoadTime = () => {
    const timeSpent = (new Date().getTime() - startTime.getTime()) * 1.000001;
    const pageLoadTimeData = `{
      "details": {
        "pageLoadTime": ${timeSpent}
      },
      "UserDetails": { "userId": "${id}", "sessionId": "${id}"}
    }`;
    const putProjectEventsRequest = {
      project: 'EvidentlySampleApp',
      events: [
        {
          timestamp: new Date(),
          type: 'aws.evidently.custom',
          data: JSON.parse(pageLoadTimeData)
        },
      ],
    };
    client.putProjectEvents(putProjectEventsRequest).promise();
  }
  return (
    !isLoading? (
    <AuthProvider>
      <CommonProvider>
        <ProductsProvider>
          <CartProvider>
            <CheckoutProvider>
              <Router>
                <Switch>
                  <RouteWrapper
                    path="/"
                    exact
                    component={() => <HomePage showDiscount={showDiscount}/>}
```

```
                    layout={CommonLayout}
                  />
                  <RouteWrapper
                    path="/checkout"
                    component={CheckoutPage}
                    layout={CommonLayout}
                  />
                  <RouteWrapper
                    path="/auth"
                    component={AuthPage}
                    layout={AuthLayout}
                  />
                </Switch>
              </Router>
            </CheckoutProvider>
          </CartProvider>
        </ProductsProvider>
      </CommonProvider>
    </AuthProvider> ) : (
      <Spinner animation="border" />
    )
  );
};

export default App;
```

Each time a user visits the sample app, a custom metric is sent to Evidently for analysis. Evidently analyzes each metric and displays results in real time on the Evidently dashboard. The following example shows a metric payload:


```
[ {"timestamp": 1637368646.468, "type": "aws.evidently.custom", "data": "{\"details
\":{\"pageLoadTime\":2058.002058},\"userDetails\":{\"userId\":\"1637368644430\",
\"sessionId\":\"1637368644430\"}}" } ]
```


Step 5: Create the project, feature, and experiment

Next, you create the project, feature, and experiment in the CloudWatch Evidently console.

To create the project, feature, and experiment for this tutorial

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Application Signals**, **Evidently**.

3. Choose **Create project** and fill out the fields. You must use **`EvidentlySampleApp`** for the project name for the sample to work correctly. For **Evaluation event storage**, choose **Don't store Evaluation events**.

   After filling out the fields, choose **Create Project**.

   For more details, see [Create a new project](#).

4. After the project is created, create a feature in that project. Name the feature **`showDiscount`**. In this feature, create two variations of the **`Boolean`** type. Name the first variation **`disable`** with a value of **`False`** and name the second variation **`enable`** with a value of **`True`**.

   For more information about creating a feature, see [Add a feature to a project](#).

5. After you have finished creating the feature, create an experiment in the project. Name the experiment **`pageLoadTime`**.

   This experiment will use a custom metric called `pageLoadTime` that measures the page load time of the page being tested. Custom metrics for experiments are created using Amazon EventBridge. For more information about EventBridge, see [What Is Amazon EventBridge?](#).

   To create that custom metric, do the following when you create the experiment:

   - Under **Metrics**, for **Metric source**, choose **Custom metrics**.
   - For **Metric name**, enter **`pageLoadTime`**.
   - For **Goal** choose **Decrease**. This indicates that we want a lower value of this metric to indicate the best variation of the feature.
   - For **Metric rule**, enter the following:
     - For **Entity ID**, enter **`UserDetails.userId`**.
     - For **Value key**, enter **`details.pageLoadTime`**.
     - For **Units**, enter **`ms`**.
   - Choose **Add metric**.

   For **Audiences**, select **100%** so that all users are entered in the experiment. Set up the traffic split between the variations to be 50% each.

   Then, choose **Create experiment** to create the experiment. After you create it, it does not start until you tell Evidently to start it.

Step 6: Start the experiment and test CloudWatch Evidently

The final steps are starting the experiment and starting the sample app.

To start the tutorial experiment

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Application Signals**, **Evidently**.
3. Choose the **EvidentlySampleApp** project.
4. Choose the **Experiments** tab.
5. Choose the button next to **pageLoadTime** and choose **Actions**, **Start experiment**.
6. Choose a time for the experiment to end.
7. Choose **Start experiment**.

   The experiment starts immediately.

Next, start the Evidently sample app with the following command:

```
npm install -f && npm start
```

Once the app has started, you will be assigned to one of the two feature variations being tested. One variation displays "20% discount" and the other doesn't. Keep refreshing the page to see the different variations.

> **Note**
>
> Evidently has sticky evaluations. Feature evaluations are deterministic, meaning for the same `entityId` and feature, a user will always receive the same variation assignment. The only time variation assignments change is when an entity is added to an override or experiment traffic is dialed up.
> However, to make the use of the sample app tutorial easy for you, Evidently reassigns the sample app feature evaluation every time that you refresh the page, so that you can experience both variations without having to add overrides.

Troubleshooting

We recommend that you use `npm` version 6.14.14. If you see any errors about building or starting the sample app and you are using a different version of npm, do the following.

To install npm version 6.14.14

1. Use a browser to connect to https://nodejs.org/download/release/v14.17.5/.
2. Download [node-v14.17.5.pkg](https://nodejs.org/download/release/v14.17.5/node-v14.17.5.pkg) and run this pkg to install npm.

   If you see a `webpack not found` error, navigate to the `evidently-sample-shopping-app` folder and try the following:

   a. Delete `package-lock.json`
   b. Delete `yarn-lock.json`
   c. Delete `node_modules`
   d. Delete the webpack dependency from `package.json`
   e. Run the following:

      ```
      npm install -f && npm
      ```

Amazon Q Developer operational investigations (Preview)

> **Note**
>
> The Amazon Q Developer operational investigations feature is in preview release and is subject to change. It is currently available in the following Regions:
>
> - US East (N. Virginia)
> - US East (Ohio)
> - US West (Oregon)
> - Asia Pacific (Hong Kong)
> - Asia Pacific (Mumbai)
> - Asia Pacific (Singapore)
> - Asia Pacific (Sydney)
> - Asia Pacific (Tokyo)
> - Europe (Frankfurt)
> - Europe (Ireland)
> - Europe (Spain)
> - Europe (Stockholm)

The Amazon Q Developer operational investigations feature is a generative AI-powered assistant that can help you respond to incidents in your system. It uses generative AI to scan your system's telemetry and quickly surface suggestions that might be related to your issue. These suggestions include metrics, logs, deployment events, and root-cause hypotheses. For a complete list of types of data that the AI assistant can surface, see Insights that Amazon Q Developer can surface in investigations

For each suggestion, you decide whether to add it to the investigation findings or to discard it. This helps Amazon Q Developer refine and iterate toward the root cause of the issue. Amazon Q Developer can help you find the root cause without having to manually identify and query multiple metrics and other sources of telemetry and events. A troubleshooting issue that would have taken hours of searching and switching between different consoles can be solved in a much shorter time.

You can create investigations in three ways:

- From within many Amazon consoles. For example, you can start an investigation when viewing a CloudWatch metric or alarm in the CloudWatch console, or from a Lambda function's **Monitor** tab on its properties page.
- By asking a question in Amazon Q chat. The question could be something like "Why is my Lambda function slow today?" or "What's wrong with my database?"
- By configuring a CloudWatch alarm action to automatically start an investigation when the alarm goes into ALARM state.

After you start an investigation with any of these methods, Amazon Q Developer scans your system to find telemetry that might be relevant to the situation, and also generates hypotheses based on what it finds. Amazon Q Developer surfaces both the telemetry data and the hypotheses, and enables you to accept or discard each one.

> **Important**
>
> To help Amazon Q Developer operational investigations (Preview) provide the most relevant information, we might use certain content from Amazon Q, including but not limited to questions that you ask Amazon Q and its response, insights, user interactions, telemetry, and metadata for service improvements. Your trust and privacy, as well as the security of your content, is our highest priority. For more information, see **Amazon Service Terms** and **Amazon responsible AI policy**.
> You can opt out of having your content collected to develop or improve the quality of Amazon Q Developer operational investigations (Preview) by creating an AI service opt-out policy for either Amazon Q or CloudWatch. For more information, see AI services opt-out policies in the Amazon Organizations User Guide.

How investigations find data for suggestions

Investigations use a wide range of data sources to determine dependency relationships and plan analysis paths, including telemetry data configurations, service configurations, and observed relationships. These dependency relationships are found more easily if you use CloudWatch Application Signals and Amazon X-Ray. When Application Signals and X-Ray aren't available, Amazon Q Developer will attempt to infer dependency relationships through co-occurring telemetry anomalies.

While Amazon Q Developer will continue to analyze telemetry data and provide suggestions without these features enabled, in order to ensure optimal quality and performance for Amazon Q Developer operational investigations, we strongly recommend that you enable the services and features listed in (Recommended) Best practices to enhance investigations.

Costs associated with Amazon Q Developer operational investigations

The Amazon Q Developer operational investigations feature is provided at no additional cost while in Preview release. During investigations, Amazon Q Developer might incur Amazon service usage including telemetry and resource queries and other API usage. While the majority of these will not be charged to your Amazon bill, there are exceptions including but not limited to CloudWatch APIs (`ListMetrics`, `GetDashboard`, `ListDashboards`, and `GetInsightRuleReport`), X-Ray APIs (`GetServiceGraph`, `GetTraceSummaries`, and `BatchGetTraces`). Amazon Q Developer also uses Amazon Cloud Control APIs which might incur usage of Amazon services such as Amazon Kinesis Data Streams and Amazon Lambda. Additionally, integration with Amazon Q Developer in chat applications which might incur usage of Amazon Simple Notification Service. For usage of these services exceeding the Amazon Free Tier, you will see charges on your Amazon bill. These charges are expected to be minimal for normal usage of Amazon Q Developer operational investigations. For more information, see Amazon Kinesis Data Streams pricing, Amazon Lambda pricing for Automation, and Amazon Simple Notification Service pricing.

Topics

- Amazon services where investigations are supported
- Getting started
- Investigate operational issues in your environment
- Manage your current investigations
- Security in operational investigations
- Troubleshooting
- Integrations with other systems
- (Recommended) Best practices to enhance investigations
- Operational investigation data retention
- Insights that Amazon Q Developer can surface in investigations

Amazon services where investigations are supported

Note

The Amazon Q Developer operational investigations feature is in preview release and is subject to change. It is currently available in the following Regions:

- US East (N. Virginia)
- US East (Ohio)
- US West (Oregon)
- Asia Pacific (Hong Kong)
- Asia Pacific (Mumbai)
- Asia Pacific (Singapore)
- Asia Pacific (Sydney)
- Asia Pacific (Tokyo)
- Europe (Frankfurt)
- Europe (Ireland)
- Europe (Spain)
- Europe (Stockholm)

You can launch investigations from telemetry data (such as CloudWatch metrics, alarms, and logs), review generated anomaly signals, and explore hypotheses on investigations. Amazon Q Developer operational investigations work best when helping you with automated troubleshooting guidance on the Amazon services listed below:

- Amazon EC2[1]
- Amazon ECS on Amazon EC2[2]
- Amazon ECS on Fargate[2]
- Amazon EKS[2]
- Amazon DynamoDB
- Amazon S3
- Amazon EBS[1]

- Lambda
- Amazon Kinesis Data Streams
- Amazon Data Firehose
- Amazon API Gateway
- Amazon SQS
- Amazon SNS

The list of services will continue to be expanded over time. Amazon Q Developer operational investigations utilizes a wide range of data sources to determine dependency relationships and plan analysis paths, including telemetry data configurations, service configurations, and observed relationships through CloudWatch Application Signals and X-Ray. Where none of the above is available, Amazon Q Developer operational investigations will attempt to infer dependency relationships through co-occurring telemetry anomalies.

Best practice setup

While Amazon Q Developer operational investigations will continue to analyze telemetry data and provide suggestions without the following features enabled, in order to ensure optimal quality and performance for Amazon Q Developer operational investigations, we highly recommend to complete the following steps:

- [1]For both Amazon EC2 and Amazon EBS, update your CloudWatch agent to version 1.30049.1 or later. For more information, see Collect metrics, logs, and traces with the CloudWatch agent.
- [2]For both Amazon ECS and Amazon EKS, enable CloudWatch Container Insights. For more information, see Container Insights.
- We recommend that you enable CloudWatch Application Signals and X-Ray. For more information, see Application Signals and **What is Amazon X-Ray**.

Getting started

Note

The Amazon Q Developer operational investigations feature is in preview release and is subject to change. It is currently available in the following Regions:

- US East (N. Virginia)

- US East (Ohio)
- US West (Oregon)
- Asia Pacific (Hong Kong)
- Asia Pacific (Mumbai)
- Asia Pacific (Singapore)
- Asia Pacific (Sydney)
- Asia Pacific (Tokyo)
- Europe (Frankfurt)
- Europe (Ireland)
- Europe (Spain)
- Europe (Stockholm)

To set up Amazon Q Developer operational investigations, you create an investigation group. You can also see a sample investigation to get an overall idea of how they work.

Topics

- See a sample investigation
- Set up operational investigations

See a sample investigation

If you'd like to see the Amazon Q Developer operational investigations feature in action before you configure it for your account, you can walk through a sample investigation. The sample investigation doesn't use your data and doesn't make data calls or start API operations in your account.

To view the sample investigation

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the left navigation pane, choose **AI Operations**, **Overview**.
3. Choose **Try a sample investigation**.

   The console displays the sample investigation, with suggestions and findings in the right pane. In each popup, choose **Next** to advance to the next part of the sample walkthrough.

Set up operational investigations

To set up Amazon Q Developer operational investigations in your account, you create an *investigation group*. Creating an investigation group is a one-time setup task. Settings in the investigation group help you centrally manage the common properties of your investigations, such as the following:

- Who can access the investigations
- Whether investigation data is encrypted with a customer managed Amazon Key Management Service key.
- How long investigations and their data are retained by default.

Currently, you can have one investigation group per account. Each investigation in your account will be part of this investigation group.

To create an investigation group and set up Amazon Q Developer operational investigations, you must be signed in to an IAM principal that has the either the **AIOpsConsoleAdminPolicy** or the **AdministratorAccess** IAM policy attached, or to an account that has similar permissions.

> **Note**
>
> To be able to choose the recommended option of creating a new IAM role for Amazon Q Developer operational investigations, you must be signed in to an IAM principal that has the `iam:CreateRole`, `iam:AttachRolePolicy`, and `iam:PutRolePolicy` permissions.

> **Important**
>
> Amazon Q Developer operational investigations uses *cross-Region inference* to distribute traffic across different Amazon Regions. For more information, see Cross-Region inference.

To create an investigation group and enable Amazon Q Developer operational investigations in your account

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the left navigation pane, choose **AI Operations**, **Investigations**.

3. Choose **Configure for this account**.
4. Enter a name for the investigation group.
5. Optionally change the retention period for investigations. For more information about what the retention period governs, see Operational investigation data retention.
6. (Optional) To encrypt your investigation data with a customer managed Amazon KMS key, choose **Customize encryption settings** and follow the steps to create or specify a key to use. If you don't specify a customer managed key, Amazon Q Developer operational investigations uses an Amazon owned key for encryption. For more information, see Encryption of investigation data.
7. If you haven't already done so, use the IAM console to provision access for your users to be able to see and manage investigations. We provide IAM roles for administrators, operators, and viewers. For more information, see User permissions.
8. (Optional) In the US East (N. Virginia) Region, you can have investigations attribute investigation actions such as adding a suggestion to the **Feed** to individual users. You do this by integrating Amazon Q Developer operational investigations with IAM Identity Center. To do so, validate that you meet the pre-requisites and then choose to allow the creation of a managed IAM Identity Center application for Amazon Q Developer operational investigations. For more information about the pre-requisites, see Amazon IAM Identity Center.
9. For **Amazon Q Developer permissions**, choose one of the following. For more information about these options, see How to control what data Amazon Q Developer has access to during investigations.

   To be able to choose either of the first two options, you must be signed in to an IAM principal that has the `iam:CreateRole`, `iam:AttachRolePolicy`, and `iam:PutRolePolicy` permissions.

   - The recommended option is to choose **Auto-create a new role with default investigation permissions**. If you choose this option, the assistant is granted the **AIOpsAssistantPolicy** IAM policy. For more information about the contents of this policy, see IAM policy for Amazon Q Developer operational investigations (AIOpsAssistantPolicy).
   - Choose **Create a new role from Amazon policy templates** to customize the permissions that Amazon Q Developer will have during investigations. If you choose this option, you must be sure to scope down the policy to only the permissions that you want Amazon Q Developer to have during investigations.
   - Choose **Assign an existing role** if you already have a role with the permissions that you want to use.

If you choose this option, you must make sure the role includes a trust policy that names `aiops.amazonaws.com` as the service principal. For more information about using service principals in trust policies, see [Amazon service principals]

We also recommend that you include a `Condition` section with the account number, to prevent a confused deputy situation. The following example trust policy illustrates both the service principal and the `Condition` section.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "aiops.amazonaws.com"
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "123456789012"
                },
                "ArnLike": {
                    "aws:SourceArn": "arn:aws:aiops:us-east-1:123456789012:*"
                }
            }
        }
    ]
}
```

10. (Optional) For **Enhanced integrations**, choose to allow Amazon Q Developer access to additional services in your system, to enable it to gather more data and be more useful.

    a. In the **Tags for application boundary detection** section, enter the existing custom tag keys for custom applications in your system. Resource tags help Amazon Q Developer narrow the search space when it is unable to discover definite relationships between resources. For example, to discover that an Amazon ECS service depends on an Amazon RDS database, Amazon Q Developer can discover this relationship using data sources such as X-Ray and CloudWatch Application Signals. However, if you haven't deployed these features, Amazon Q Developer will attempt to identify possible relationships. Tag boundaries can be used to narrow the resources that will be discovered by Amazon Q Developer in these cases.

You don't need to enter tags created by myApplications or Amazon CloudFormation, because Amazon Q Developer can automatically detect those tags.

b. CloudTrail records events about changes in your system including deployment events. These events can often be useful to Amazon Q Developer to create hypotheses about root causes of issues in your system. In the **CloudTrail for change event detection** section, you can do one or both of the following.

   - Give Amazon Q Developer some access to the events logged by Amazon CloudTrail by enabling **Allow the assistant access to CloudTrail change events through the CloudTrail Event history**. For more information, see Working with CloudTrail Event history.
   - Give Amazon Q Developer access to additional CloudTrail data by entering one or more CloudTrail trails, which are records of activities within your Amazon account. This is supported only for trails that are sent to log groups in CloudWatch Logs. For more information about trails see Working with CloudTrail trails.

c. The **X-Ray for topology mapping** and **Application Signals for health assessment** sections point out other Amazon services that can help Amazon Q Developer find information. If you have deployed them and you have granted the **AIOpsAssistantPolicy** IAM policy to Amazon Q Developer, it will be able to access X-Ray and Application Signals telemetry.

   For more information about how these services help Amazon Q Developer, see X-Ray and CloudWatch Application Signals

11. (Optional) If you are in the US East (N. Virginia) Region, you can integrate Amazon Q Developer operational investigations with a third-party ticketing system. Integrating with a ticketing tool enables Amazon Q Developer to send information about an investigation to that ticketing tool. This feature is available only in US East (N. Virginia).

> **Important**
>
> When you create an integration with a third-party ticketing system, the system creates a secret in Amazon Secrets Manager. This secret contains your basic authentication credentials for the third-party tool and is used to connect your Amazon Web Services

account to that tool. If you delete the integration, the secret that contains your authentication credentials is also deleted.

To integrate Amazon Q Developer operational investigations with a third-party ticketing tool, do the following in the **Third-party integrations** area:

a. Choose **Add integration**.

b. In the **Add integration** dialog box, do the following:

   i. For **Name**, enter a name to identify this integration in your investigations.

   ii. For **Instance type**, choose from the available third-party tools, such as Jira or ServiceNow.

   > **Note**
   >
   > Integration with Jira is supported only for Jira Cloud.

   iii. Provide the additional information required to integrate with your selected tool:

   Jira

   - **Username** – A valid email address with access to the project being onboarded.
   - **Password** – Your API token value from Atlassian. For more information, see Manage API tokens for your Atlassian account on the Atlassian website.
   - **Jira site name** – The application name for your Jira project. This is typically the first component of the URL for your project in Atlassian. For example, in the URL `https://AnyCompany.atlassian.net`, the application name is `AnyCompany`. If you are uncertain, contact your Jira project manager to verify this information.
   - **Project key** – The project key for your Jira project.

     To retrieve this project key, sign in to your Jira project, choose **Administration, Project, View all projects**, and then copy the **Key** value from the appropriate project.

> **Important**
>
> Ensure that your Jira account and email address both have access to the project. If you are uncertain, contact your Jira project manager to verify this information.

ServiceNow

- **Username** – Your ServiceNow username.
- **Password** – Your ServiceNow instance password.
- **Instance ID** – Your ServiceNow instance ID.

You can view this information by signing in to your ServiceNow account. If you are uncertain, contact your ServiceNow administrator to verify this information.

c. Choose **Connect**.

> **Important**
>
> After you have completed this configuration procedure, we recommend that you create a test investigation and try out the integration before using it in an active investigation.

12. (Optional) You can integrate Amazon Q Developer operational investigations with a *chat channel* using Amazon Q Developer in chat applications. This makes it possible to receive notifications about an investigation through the chat channel. Amazon Q Developer operational investigations and Amazon Q Developer in chat applications support chat channels in the following applications:

    - Slack
    - Microsoft Teams

If you want to integrate with a chat channel, we recommend that you complete some other steps before performing this step in the create an investigation group process. For more information, see Integration with third-party chat systems.

To then perform the steps here to integrate with a chat channel in Amazon Q Developer in chat applications, do the following:

a. In the **Chat client integration** section, choose **Select SNS topic**.

b. Select the SNS topic to use for sending notifications about your investigations.

13. Choose **Complete setup**.

Investigate operational issues in your environment

Note

The Amazon Q Developer operational investigations feature is in preview release and is subject to change. It is currently available in the following Regions:

- US East (N. Virginia)
- US East (Ohio)
- US West (Oregon)
- Asia Pacific (Hong Kong)
- Asia Pacific (Mumbai)
- Asia Pacific (Singapore)
- Asia Pacific (Sydney)
- Asia Pacific (Tokyo)
- Europe (Frankfurt)
- Europe (Ireland)
- Europe (Spain)
- Europe (Stockholm)

Contents

Create an investigation

Create an investigation from an Amazon console page

You can start an investigation from several Amazon consoles, including (but not limited to) CloudWatch alarm pages, CloudWatch metric pages, and Lambda monitoring pages.

To start an investigation from an Amazon console page

1. In the graph of the metric or alarm that you want to investigate, select the time range that you want the investigation to include.
2. If the top of the page has an **Investigate** button, choose it and then choose **Start new investigation**.

   Otherwise, choose the vertical ellipsis menu icon

   for the metric, and choose **Investigate**, **Start a new investigation**.
3. In the **Investigation** pane, enter a name for the investigation in **New investigation title**, and optionally enter notes about the selected metric or alarm. Then choose **Start investigation**.

   The investigation starts. Amazon Q Developer scans your telemetry data to find data that might be associated with this situation.
4. To move the investigation data to the larger pane, choose **Open in full page**.
5. For detailed instructions about steps that you can take while continuing the investigation, see View and continue an open investigation.

Create an investigation from Amazon Q chat

You can ask questions about issues in your deployment in Amazon Q Developer chat. The question could be something like "Why is my Lambda function slow today?"

When you do so, Amazon Q Developer might ask follow up questions and run a health check regarding the issue. After the health check, the chat will prompt you about whether you want to start an investigation.

For more information and more sample questions, see Chatting with Amazon Q Developer about Amazon..

For detailed instructions about steps that you can take while continuing the investigation after it has been started, see View and continue an open investigation.

Create an investigation from a CloudWatch alarm action

When you create a CloudWatch alarm, you can specify for it to automatically start an investigation when it goes into ALARM state. You can do this for both metric alarms and composite alarms. For more information about creating alarms, see Alarming on metrics and Create a composite alarm.

View and continue an open investigation

Use the steps in this section to view and continue and existing investigation

To view and continue an investigation

1. If you aren't already on the page for the investigation, do the following:

   a. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.

   b. In the left navigation pane, choose **AI Operations**, **Investigations**.

   c. Choose the name of the investigation.

   > **Tip**
   >
   > If your investigation is in US East (N. Virginia) and you have integrated Amazon Q Developer operational investigations with ServiceNow or Jira, you can associate the investigation with the integrated tool from the investigations heading. For example, for Jira integrations, hover over **Jira Cloud** in the investigation header, and then choose from **Add existing ticket**, **Create new ticket**, or **Edit integration**.

> If no integration has been set up yet, you can configure one. For information, see step 3 in [Set up operational investigations](#).

2. The **Feed** section displays the items that have been added to the investigation findings, including the metric or alarm that was originally selected to start the investigation with.

   The pane on the right includes tabs. Choose the **Suggestions** tab.

3. The **Suggestions** tab displays *observations* of other telemetry that Amazon Q Developer has found that might be related to the investigation. It might also include *hypotheses*, which are possible reasons or root causes that Amazon Q Developer has found for the situation.

   Both observations and hypotheses are written in natural language by Amazon Q Developer.

   You have several options:

   - For each suggestion, you can choose **Accept** or **Discard**.

     When you choose **Accept**, the suggestion is added to the **Feed** section, and Amazon Q Developer uses this information to direct further scanning and suggestions.

     If you choose **Discard**, the suggestion is moved to the **Discarded** tab.

   - For each observation-type suggestion, you can choose to expand the graph in the **Suggestions** tab, or open it in the CloudWatch console to see more details about it.

   - Some of the observations might be results of CloudWatch Logs Insights queries that Amazon Q Developer ran as part of the investigation. When an observation is a CloudWatch Logs Insights query result, the query itself is displayed as part of the observation. You can edit the query and re-run it. To do so, choose the vertical ellipsis menu icon

     by the results, and then choose **Open in Logs Insights**. For more information, see [Analyzing log data with CloudWatch Logs Insights](#).

   - If you know of telemetry in an Amazon service that might apply to this investigation, you can go to that service's console and add the telemetry to the investigation. For example, to add a Lambda metric to the investigation, you can do the following:

     1. Open the Lambda console.
     2. In the **Monitor** section, find the metric.

3. Open the vertical ellipsis context menu

⋮

for the metric, choose **Investigate**, **Add to investigation** Then, in the **Investigate** pane, select the name of the investigation.

- When you view a hypothesis in the **Suggestions** tab, you can choose **Show reasoning** to display the data that Amazon Q Developer used to generate the hypothesis.
- If your investigation is in the US East (N. Virginia) Region and you have integrated Amazon Q Developer operational investigations with a third-party ticketing system, you can attach this investigation to a ticket in that system. To do so, hover on **No ticket attached** above the **Feed**, and then choose either **Attach ticket** or **Create ticket**.
- You can choose the **Discarded** tab and view the suggestions that have been previously discarded. To add one of them to the findings, choose **Restore to findings**.
- To add notes to the findings, choose **New note** in the **Feed** pane. Then enter your notes and choose **Add**.

4. When you add a hypothesis to the **Feed** area, it might display **Show suggested actions**. If so, choosing this displays possible actions that you can take, assuming that hypothesis is correct about the issue. Possible actions include the following:

   - **Documentation suggestions** are links to Amazon documentation that can help you understand the issue that you are working on, and how to solve it. To view suggested documentation, choose its **Review** link
   - **Runbook suggestions** are suggestions that leverage the pre-defined *runbooks* in Systems Manager Automation. Each runbook defines a number of steps for performing a task on an Amazon resource.

   > **Important**
   >
   > There is a charge for executing an Automation runbook. However, Amazon Q Developer operational investigations provides you with a preview of actions that a suggested runbook takes, giving you an opportunity to better evaluate whether to execute the runbook. For information about Automation pricing, see [Amazon Systems Manager pricing for Automation](#).

For information about continuing with a runbook action, see [Reviewing and executing suggested runbook remediations for Amazon Q Developer operational investigations]() before continuing with the following step in this procedure.

5. To end an investigation, choose **End investigation** and then optionally add final notes. Then choose **Save**.

   The investigation status changes to **Archived**. You can restart archived investigations by opening the investigation page and choosing **Restart investigation**.

   We recommend that you don't leave investigations open indefinitely, because alarm state transitions related to the investigation will keep being added to the investigation as long as it is open.

> **Note**
>
> At some points, you might see **Completed the analysis. Finished with the investigation.** displayed above the **Feed** area. If you then add more telemetry to the findings, this message changes and Amazon Q Developer begins scanning your telemetry again, based on the new data that you added to the findings.

Reviewing and executing suggested runbook remediations for Amazon Q Developer operational investigations

When you add a hypothesis to the **Feed** area of an active investigation, Amazon Q Developer operational investigations might display **Show suggested actions**. One suggested action might be to view documentation with information to help you remediate a problem manually.

Another suggestion might be to use an *Automation runbook* to attempt to automatically resolve the issue. Automation is a capability in Systems Manager, another Amazon Web Services service. Automation runbooks define a series of steps, or actions, to be run on the resources that you select. Each runbook is designed to address a specific issue. Runbooks can address a variety of operational needs: Creating, repairing, reconfiguring, installing, troubleshooting, remediating, duplicating, and more. For more information about Automation, see [Integration with Amazon Systems Manager Automation]().

Before you begin

Before working with Automation runbooks in an investigation, be aware of the following important considerations:

- Choosing to execute a runbook incurs charges. For information, see Amazon Systems Manager pricing.
- Root causes and runbook suggestions are powered by automated reasoning and generative artificial intelligence services.

> **Important**
>
> You are responsible for actions that result from executing runbook steps and the choice of parameter values entered during runbook execution. You might need to edit the suggested runbook to make sure the runbook performs as expected. For more information, see **Amazon responsible AI policy**.

- Depending on the runbook, you might need to enter values for the runbook's **Input parameters** before the execution can run.
- The runbook executes using the IAM permissions assigned to the operator. If necessary, sign in with different IAM permissions to execute the runbook. In addition to permissions for the actions being taken, you'll need additional Systems Manager permissions to execute runbook steps. For more information, see Setting up Automation in the *Amazon Systems Manager User Guide*.

To review and execute suggested runbook actions for Amazon Q Developer operational investigations

1. To view information about a suggested runbook, choose **Review** for information about how to execute the runbook steps.

   On the investigation details page, choose **Suggestions**.

2. In the **Suggestions** pane, review the list of hypotheses based on the system's analysis of the issue under investigation.

   For each hypothesis, you can choose from the following options:

   - **Show reasoning** – View more information about why the system has generated the hypothesis.

- **View actions** – View the suggested actions for the issue. Not all hypotheses will include suggested actions.
- **Accept** – Accept the hypothesis and add it to the investigation's **Feed** section.

> **Note**
>
> Accepting the hypothesis doesn't automatically run the associated runbook solution. You can view suggested runbooks before accepting a hypothesis, but you must accept the hypothesisto execute a runbook.

- **Discard** – Reject the hypothesis and don't engage with it any further.

3. After you choose **View action**, in the **Suggested actions** pane, review the list of suggested actions you can take to address the issue. Suggested actions can include one or more of the following:

   - **Amazon knowledge articles** – Provides information about steps you can take to manually address the issue, plus a link to more information.
   - **Amazon documentation** – Provides links to user documentation topics related to the issue.
   - **Amazon-owned runbooks** – Lists one or more Automation runbooks that are managed by Amazon that you can run to attempt issue resolution.
   - **Runbooks owned by you** – Lists one or more custom Automation runbooks created by you or someone else in your account or organization, which you can run to attempt issue resolution.

   > **Note**
   >
   > The system automatically generates this list of runbooks by evaluating keywords in your custom runbooks and then comparing them to terms related to the issue being investigated.
   > More keyword matches mean a particular custom runbook appears higher in the **Runbooks owned by you** list.

4. After reviewing the hypothesis, you can examine a specific suggested action further and read related documentation by choosing **Learn more**. You can also choose **Review details** to inspect suggested runbooks owned by Amazon and you.
5. When choosing **Review details** for runbooks, do the following:

a. For **Runbook description**, review the content, which provides an overview of the actions the runbook can take to remediate the issue being investigated. Choose **View steps** to visualize the runbook's workflow and drill into the details of individual steps.

b. For **Input parameters**, specify values for any parameters required by the runbook. These parameters vary from runbook to runbook.

c. For **Execution preview**, carefully review the information. This information explains what the scope and impact would be if you choose to execute the runbook.

   The **Execution preview** content provides the following information:

   - How many accounts and Regions the runbook operation would occur in.
   - The types of actions that would be taken, and how many of each type.

     Action types include the following:

     - `Mutating`: A runbook step would make changes to the targets through actions that create, modify, or delete resources.
     - `Non-Mutating`: A runbook step would retrieve data about resources but not make changes to them. This category generally includes `Describe`, `List`, `Get`, and similar read-only API actions.
     - `Undetermined`: An undetermined step invokes executions performed by another orchestration service like Amazon Lambda, Amazon Step Functions, or Run Command, a capability of Amazon Systems Manager. An undetermined step might also call a third-party API or run a Python or PowerShell script. Systems Manager Automation can't detect what the outcome would be of the orchestration processes or third-party API executions, and therefore can't evaluate them. The results of those steps would have to be manually reviewed to determine their impact.

     For information about supported actions and their impact types, see Remediation impact types of runbook actions in the *Amazon Systems Manager User Guide*.

d. Review the preview information carefully before deciding whether to proceed.

   At this point, you can choose one of the following actions:

   - Stop and do not execute the runbook.
   - Change the input parameters before executing the runbook.
   - Execute the runbook with the options you have already selected.

> **Important**
>
> Choosing to execute the runbook incurs charges. For information, see Amazon Systems Manager pricing.

6. If you want to execute the runbook, choose **Execute**.

   If you already accepted the hypothesis, the execution runs.

   If you have not already accepted the hypothesis, a dialog box prompts you to accept it before the execution runs.

After you choose **Execute** for a runbook, that action is added to the **Feed** pane of the investigation. From the investigation, you can monitor new data in the metrics in the findings to see if the runbook actions are correcting the issue.

Manage your current investigations

> **Note**
>
> The Amazon Q Developer operational investigations feature is in preview release and is subject to change. It is currently available in the following Regions:
>
> - US East (N. Virginia)
> - US East (Ohio)
> - US West (Oregon)
> - Asia Pacific (Hong Kong)
> - Asia Pacific (Mumbai)
> - Asia Pacific (Singapore)
> - Asia Pacific (Sydney)
> - Asia Pacific (Tokyo)
> - Europe (Frankfurt)
> - Europe (Ireland)
> - Europe (Spain)

- Europe (Stockholm)

You can view a list of your current investigations, end active investigations, re-open archived investigations, rename, and delete investigations. You can take these actions on individual investigations, or in bulk.

To manage your current investigations

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the left navigation pane, choose **AI Operations**, **Investigations**.
3. (Optional) Filter the investigations displayed in the list by name or investigation state.
4. Select the checkboxes for the investigation or investigations that you want to take action on.
5. Choose **End investigation**, **Rename**, or **Delete**.

   You will be prompted to confirm your action or to input a new investigation title.

Restart an archived investigation

You can restart archived investigations.

To restart an archived investigation

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the left navigation pane, choose **AI Operations**, **Investigations**.
3. Choose the name of an archived investigation.
4. Choose **Restart investigation**.
5. For instructions for working in an existing investigation, see View and continue an open investigation.

Security in operational investigations

Note

The Amazon Q Developer operational investigations feature is in preview release and is subject to change. It is currently available in the following Regions:

- US East (N. Virginia)
- US East (Ohio)
- US West (Oregon)
- Asia Pacific (Hong Kong)
- Asia Pacific (Mumbai)
- Asia Pacific (Singapore)
- Asia Pacific (Sydney)
- Asia Pacific (Tokyo)
- Europe (Frankfurt)
- Europe (Ireland)
- Europe (Spain)
- Europe (Stockholm)

This section includes topics about how Amazon Q Developer operational investigations integrate with Amazon security and permissions features.

Topics

User permissions

Amazon has created three managed IAM policies that you can use for your users who will be working with Amazon Q Developer operational investigations.

- AIOpsConsoleAdminPolicy– grants an administrator the ability to set up Amazon Q Developer operational investigations in the account, access to Amazon Q Developer operational investigations actions, the management of trusted identity propagation, and the management of integration with IAM Identity Center and organizational access.

- AIOpsOperatorAccess– grants a user access to investigation actions including starting an investigation. It also grants additional permissions that are necessary for accessing investigation events.
- AIOpsReadOnlyAccess– grants read-only permissions for Amazon Q Developer operational investigations and other related services.

We recommend that you use three IAM principals, granting one of them the **AIOpsConsoleAdminPolicy** IAM policy, granting another the **AIOpsOperatorAccess** policy, and granting the third the **AIOpsReadOnlyAccess** policy. These principals could be either IAM roles (recommended) or IAM users. Then your users who work with Amazon Q Developer operational investigations would sign on with one of these principals.

How to control what data Amazon Q Developer has access to during investigations

When you enable the Amazon Q Developer operational investigations feature, you specify what permissions that Amazon Q Developer has to access your resources during investigations. You do this by assigning an IAM role to the assistant.

To enable Amazon Q Developer to access resources and be able to make suggestions and hypotheses, the recommended method is to attach the **AIOpsAssistantPolicy** to the assistant's role. This grants the assistant permissions to analyze your Amazon resources during your investigations. For information about the complete contents of this policy, see IAM policy for Amazon Q Developer operational investigations (AIOpsAssistantPolicy).

You can also choose to attach the general Amazon **ReadOnlyAccess** to the assistant's role, in addition to attaching **AIOpsAssistantPolicy**. The reason to do this is that Amazon updates **ReadOnlyAccess** more frequently with permissions for new Amazon services and actions that are released. The **AIOpsAssistantPolicy** will also be updated for new actions, but not as frequently.

If you want to scope down the permissions granted to Amazon Q Developer, you can attach a custom IAM policy to the assistant's IAM role instead of attaching the **AIOpsAssistantPolicy** policy. To do this, start your custom policy with the contents of AIOpsAssistantPolicy and then remove permissions that you don't want to grant to Amazon Q Developer. This will prevent the assistant from being able to make suggestions based on the Amazon services or actions that you don't grant access to.

> **Note**
>
> Anything that Amazon Q Developer can access can be added to the investigation and seen by your investigation operators. We recommend that you align Amazon Q Developer operational investigations permissions with the permissions that your investigation group operators have.

Allowing Amazon Q Developer to decrypt encrypted data during investigations

If you encrypt your data in any of the following services with a customer managed key in Amazon KMS, and you want Amazon Q Developer to be able to decrypt the data from these services and include them in investigations, you'll need to attach one or more additional IAM policies to the assistant's IAM role.

- Amazon Step Functions

The policy statement should include a context key for encryption context to help scope down the permissions. For example, the following policy would enable the Amazon Q Developer to decrypt data for a Step Functions state machine.

```
{
  "Version": "2012-10-17",
  "Statement": [
        {
            "Sid": "AIOPSKMSAccessForStepFunctions",
            "Effect": "Allow",
            "Principal": {
                "Service": "aiops.amazonaws.com"
            },
            "Action":
            [
                "kms:Decrypt"
            ],
            "Resource": "*",
            "Condition":
            {
                "StringEquals":
                {
                     "kms:ViaService": "states.*.amazonaws.com",
```

```
                        "kms:EncryptionContext:aws:states:stateMachineArn":
 "arn:aws:states:region:accountId:stateMachine:*"
                    }
                }
            }
        ]
}
```

For more information about these types of policies and using these context keys, see kms:ViaService and kms:EncryptionContext:*context-key* in the Amazon Key Management Service Developer Guide, and aws:SourceArn in the IAM User Guide.

Encryption of investigation data

For the encryption of your investigation data, Amazon offers two options:

- **Amazon owned keys**– By default, Amazon Q Developer encrypts investigation data at rest with an Amazon owned key. You can't view or manage Amazon owned keys, and you can't use them for other purposes or audit their use. However, you don't have to take any action or change any settings to use these keys. For more information about Amazon owned keys, see Amazon owned keys.
- **Customer managed keys**– These are keys that you create and manage yourself. You can choose to use a customer managed key instead of an Amazon owned key for your investigation data. For more information about customer managed keys, see Customer managed keys.

> **Note**
>
> Amazon Q Developer automatically enables encryption at rest using Amazon owned keys at no charge. If you use a customer managed key, Amazon KMS charges apply. For more information about pricing, see Amazon Key Management Service pricing.

For more information about Amazon KMS, see Amazon Key Management Service.

Using a customer managed key for your investigation group

You can associate an investigation group with a customer managed key, and then all investigations created in that group will use the customer managed key to encrypt your investigation data at rest.

Amazon Q Developer operational investigations customer managed key usage has the following conditions:

- Amazon Q Developer operational investigations supports only symmetric encryption Amazon KMS keys with the default key spec, `SYMMETRIC_DEFAULT`, and that have usage defined as `ENCRYPT_DECRYPT`.
- For a user to create or update an investigation group with a customer managed key, that user must have the `kms:DescribeKey`, `kms:GenerateDataKey`, and `kms:Decrypt` permissions.
- For a user to create or update an investigation in an investigation group that uses a customer managed key, that user must have the `kms:GenerateDataKey` and `kms:Decrypt` permissions.
- For a user to view investigation data in an investigation group that uses a customer managed key, that user must have the `kms:Decrypt` permission.

Setting up investigations to use a Amazon KMS customer managed key

First, if you don't already have a symmetric key that you want to use, create a new key with the following command.

```
aws kms create-key
```

The command output includes the key ID and the Amazon Resource Name (ARN) of the key. You will need those in later steps in this section. The following is an example of this output.

```
{
"KeyMetadata": {
"Origin": "AWS_KMS",
        "KeyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
        "Description": "",
        "KeyManager": "CUSTOMER",
        "Enabled": true,
        "CustomerMasterKeySpec": "SYMMETRIC_DEFAULT",
        "KeyUsage": "ENCRYPT_DECRYPT",
        "KeyState": "Enabled",
        "CreationDate": 1478910250.94,
        "Arn": "arn:aws:kms:us-west-2:111122223333:key/6f815f63-e628-448c-8251-
e4EXAMPLE",
        "AWSAccountId": "111122223333",
        "EncryptionAlgorithms": [
            "SYMMETRIC_DEFAULT"
```

```
        ]
    }
}
```

Set permissions on the key

Next, set permissions on the key. By default, all Amazon KMS keys are private. Only the resource owner can use it to encrypt and decrypt data. However, the resource owner can grant permissions to access the key to other users and resources. With this step, you give the AI Operations service principal permission to use the key. This service principal must be in the same Amazon Region where the KMS key is stored.

As a best practice, we recommend that you restrict the use of the KMS key to only those Amazon accounts or resources that you specify.

The first step to set the permissions is to save the default policy for your key as `policy.json`. Use the following command to do so. Replace *`key-id`* with the ID of your key.

```
aws kms get-key-policy --key-id key-id --policy-name default --output text > ./
policy.json
```

Open the `policy.json` file in a text editor and add the following policy sections into that policy. Separate the existing statement from the new sections with a comma. These new sections use `Condition` sections to enhance the security of the Amazon KMS key. For more information, see [Amazon KMS keys and encryption context](#).

This policy provides permissions for service principals for the following reasons:

- The `aiops` service needs `GenerateDataKey` permissions to get the data key and use that data key to encrypt your data while it is stored in rest. The `Decrypt` permission is needed to decrypt your data while reading from the data store. The decryption happens when you read the data using `aiops` APIs or when you update the investigation or investigation event. The update operation fetches the existing data after decrypting it, updates the data, and stores the updated data in the data store after encrypting
- The CloudWatch alarms service can create investigations or investigation events. These create operations verify that the caller has access to the Amazon KMS key defined for the investigation group. The policy statement gives the `GenerateDataKey` and `Decrypt` permissions to the CloudWatch alarms service to create investigations on behalf of you.

> **Note**
>
> The following policy assumes that you follow the recommendation of using three IAM principals, and granting one of them the **AIOpsConsoleAdminPolicy** IAM policy, granting another the **AIOpsOperatorAccess** policy, and granting the third the **AIOpsReadOnlyAccess** policy. These principals could be either IAM roles (recommended) or IAM users. Then your users who work with Amazon Q Developer operational investigations would sign on with one of these principals.
> For the following policy, you'll need the ARNs of those three principals.

```
{
    "Sid": "Enable AI Operations Admin for the DescribeKey permissions",
    "Effect": "Allow",
    "Principal": {
        "AWS": "arn:aws:iam::{account-id}:role/{AIOpsConsoleAdmin}"
    },
    "Action": [
        "kms:DescribeKey"
    ],
    "Resource": "*",
    "Condition": {
        "StringEquals": {
            "kms:ViaService": "aiops.{region}.amazonaws.com"
        }
    }
},
{
   "Sid": "Enable AI Operations Admin and Operator for the Decrypt and GenerateDataKey
 permissions",
   "Effect": "Allow",
    "Principal": {
        "AWS": [
            "arn:aws:iam::{account-id}:role/{AIOpsConsoleAdmin}",
            "arn:aws:iam::{account-id}:role/{AIOpsOperator}"
         ]
    },
    "Action": [
        "kms:Decrypt",
        "kms:GenerateDataKey"
    ],
```

```
     "Resource": "*",
     "Condition": {
        "StringEquals": {
             "kms:ViaService": "aiops.{region}.amazonaws.com"
         },
         "ArnLike": {
             "kms:EncryptionContext:aws:aiops:investigation-group-arn": "arn:aws:aiops:
{region}:{account-id}:investigation-group/*"
         }
     }
 },
 {
    "Sid": "Enable AI Operations ReadOnly for the Decrypt permission",
    "Effect": "Allow",
     "Principal": {
         "AWS": "arn:aws:iam::{account-id}:role/{AIOpsReadOnly}"
     },
     "Action": [
         "kms:Decrypt"
     ],
     "Resource": "*",
     "Condition": {
        "StringEquals": {
             "kms:ViaService": "aiops.{region}.amazonaws.com"
         },
         "ArnLike": {
             "kms:EncryptionContext:aws:aiops:investigation-group-arn": "arn:aws:aiops:
{region}:{account-id}:investigation-group/*"
         }
     }
 },
 {
    "Sid": "Enable the AI Operations service to have the DescribeKey permission",
    "Effect": "Allow",
     "Principal": {
         "Service": "aiops.amazonaws.com"
     },
     "Action": [
         "kms:DescribeKey"
     ],
     "Resource": "*",
     "Condition": {
        "StringEquals": {
             "aws:SourceAccount": "{account-id}"
```

```
        },
        "StringLike": {
            "aws:SourceArn": "arn:aws:aiops:{region}:{account-id}:investigation-group/
*"
        }
    }
 },
 {
   "Sid": "Enable the AI Operations service to have the Decrypt and GenerateDataKey
 permissions",
   "Effect": "Allow",
    "Principal": {
        "Service": "aiops.amazonaws.com"
    },
    "Action": [
        "kms:Decrypt",
        "kms:GenerateDataKey"
    ],
    "Resource": "*",
    "Condition": {
       "StringEquals": {
            "aws:SourceAccount": "{account-id}"
        },
        "StringLike": {
            "aws:SourceArn": "arn:aws:aiops:{region}:{account-id}:investigation-group/
*"
        },
        "ArnLike": {
            "kms:EncryptionContext:aws:aiops:investigation-group-arn": "arn:aws:aiops:
{region}:{account-id}:investigation-group/*"
        }
    }
 },
 {
    "Sid": "Enable CloudWatch to have the Decrypt and GenerateDataKey permissions",
    "Effect": "Allow",
    "Principal": {
        "Service": "aiops.alarms.cloudwatch.amazonaws.com"
    },
    "Action": [
        "kms:GenerateDataKey",
        "kms:Decrypt"
    ],
    "Resource": "*",
```

```
    "Condition": {
        "ArnLike": {
            "kms:EncryptionContext:aws:aiops:investigation-group-arn": "arn:aws:aiops:
{region}:{account-id}:investigation-group/*"
        },
        "StringEquals": {
            "aws:SourceAccount": "{account-id}",
            "kms:ViaService": "aiops.{region}.amazonaws.com"
        },
        "StringLike": {
            "aws:SourceArn": "arn:aws:cloudwatch:{region}:{account-id}:alarm:*"
        }
    }
  }
```

After you've updated the policy, assign it to the key by entering the following command.

```
aws kms put-key-policy --key-id key-id --policy-name default --policy file://
policy.json
```

Associate the key with the investigation group

When you use the CloudWatch console to create an investigation group, you can choose to associate the Amazon KMS key with the investigation group. For more information, see Set up operational investigations.

You can also associate a customer managed key with an existing investigation group.

Changing your encryption configuration

You can update an investigation group to change between using a customer managed key or a service owned key. You can also change from using one customer managed key to using another. When you make such a change, the change applies to new investigations created after the change. Previous investigations are still associated with the old encryption configuration. Current ongoing investigations also continue using the original key for new data.

As long as a previously-used key is active and Amazon Q has access to it for investigations, you can retrieve the older investigations encrypted with that method, as well as data in current investigations that was encrypted with the previous key. If you delete a previously-used key or revoke access to it, the investigation data encrypted with that key can't be retrieved.

Cross-Region inference

Amazon Q Developer operational investigations uses *cross-Region inference* to distribute traffic across different Amazon Region. Although the data remains stored only in the primary Region, when using cross-Region inference, your investigation data might move outside of your primary Region. All data will be transmitted encrypted across Amazon's secure network. For more information see Cross-Region inference in the Amazon Q Developer user guide.

For details about where cross-Region inference distribution occurs for each Region, see the following table.

Supported Amazon Q Developer operational investigations geography	Investigation Region	Possible inference Regions
United States (US)	US East (N. Virginia)	US East (N. Virginia), US East (Ohio), US West (Oregon)
	US East (Ohio)	US East (N. Virginia), US East (Ohio), US West (Oregon)
	US West (Oregon)	US East (N. Virginia), US East (Ohio), US West (Oregon)
Europe (EU)	Europe (Frankfurt)	US East (N. Virginia), US East (Ohio), US West (Oregon)
	Europe (Ireland)	US East (N. Virginia), US East (Ohio), US West (Oregon)
	Europe (Spain)	US East (N. Virginia), US East (Ohio), US West (Oregon)
	Europe (Stockholm)	US East (N. Virginia), US East (Ohio), US West (Oregon)
Asia-Pacific (AP)	Asia Pacific (Hong Kong)	US East (N. Virginia), US East (Ohio), US West (Oregon)

Supported Amazon Q Developer operational investigations geography	Investigation Region	Possible inference Regions
	Asia Pacific (Mumbai)	US East (N. Virginia), US East (Ohio), US West (Oregon)
	Asia Pacific (Singapore)	US East (N. Virginia), US East (Ohio), US West (Oregon)
	Asia Pacific (Sydney)	US East (N. Virginia), US East (Ohio), US West (Oregon)
	Asia Pacific (Tokyo)	US East (N. Virginia), US East (Ohio), US West (Oregon)

Troubleshooting

Note

The Amazon Q Developer operational investigations feature is in preview release and is subject to change. It is currently available in the following Regions:

- US East (N. Virginia)
- US East (Ohio)
- US West (Oregon)
- Asia Pacific (Hong Kong)
- Asia Pacific (Mumbai)
- Asia Pacific (Singapore)
- Asia Pacific (Sydney)
- Asia Pacific (Tokyo)
- Europe (Frankfurt)
- Europe (Ireland)
- Europe (Spain)

- Europe (Stockholm)

Topics

Amazon Q Developer cannot assume the necessary IAM roles or permissions. Please verify required roles and permissions are correctly configured

Amazon Q Developer operational investigations use an IAM role to be able to access information in your topology. This IAM role must be configured with adequate permissions. For more information about the necessary permissions, see How to control what data Amazon Q Developer has access to during investigations.

Unable to identify event source. Verify that the resource exists in your application topology and the resource type is supported.

There are several Amazon services and features that we recommend you to enable to provide additional valuable information to Amazon Q Developer. These services and features can help Amazon Q Developer identify event sources. For more information, see (Recommended) Best practices to enhance investigations.

Analysis complete. Submit additional findings to receive updated suggestions

When you see this message, Amazon Q Developer has finished analyzing your topology and telemetry based on the findings that it has found so far. If you think that the root cause hasn't been found, you can manually add more telemetry to the investigation, and this might cause Amazon Q Developer to scan your system again based on the new information.

To add new telemetry, navigate to that service's console and add the telemetry to the investigation. For example, to add a Lambda metric to the investigation, you can do the following:

1. Open the Lambda console.
2. In the **Monitor** section, find the metric.
3. Open the vertical ellipsis context menu

   ⋮

   for the metric, choose **Investigate**, **Add to investigation** Then, in the **Investigate** pane, select the name of the investigation.

Integrations with other systems

> **Note**
>
> The Amazon Q Developer operational investigations feature is in preview release and is subject to change. It is currently available in the following Regions:
>
> - US East (N. Virginia)
> - US East (Ohio)
> - US West (Oregon)
> - Asia Pacific (Hong Kong)
> - Asia Pacific (Mumbai)
> - Asia Pacific (Singapore)
> - Asia Pacific (Sydney)
> - Asia Pacific (Tokyo)
> - Europe (Frankfurt)
> - Europe (Ireland)
> - Europe (Spain)
> - Europe (Stockholm)

Topics

- Integration with third-party chat systems
- Amazon IAM Identity Center

Integration with Amazon Systems Manager Automation

Amazon Q Developer operational investigations is integrated with Automation, a capability of Amazon Systems Manager. You don't need to configure integration, but you might need to update Amazon Identity and Access Management (IAM) permissions so you can use Automation runbooks.

What is Amazon Systems Manager?

Systems Manager helps you centrally view, manage, and operate *managed nodes* at scale in Amazon, on-premises, and multicloud environments. In Systems Manager, a managed node is any machine configured for use with Systems Manager. For information, see the *Amazon Systems Manager User Guide*.

What is Systems Manager Automation?

Automation performs common maintenance, deployment, troubleshooting, and remediation tasks through the use of *runbooks*. Each runbook defines a number of steps for performing tasks. Each step is associated with a particular *action*. The action determines the inputs, behavior, and outputs of the step. For descriptions of the nearly two dozen actions that are supported for runbooks, see the Systems Manager Automation actions reference in the *Amazon Systems Manager User Guide*.

Automation provides over 400 Amazon managed runbooks. For details about each runbook, including a step-by-step description of the actions performed when executed, see the Systems Manager Automation runbook reference. Customers can also design their own runbooks to address specific scenarios in their environments. For information, see Creating your own runbooks in the *Amazon Systems Manager User Guide*.

For information about working with runbooks in an investigation, see Reviewing and executing suggested runbook remediations for Amazon Q Developer operational investigations.

Third-party ticketing systems

In the US East (N. Virginia) Region, you can integrate Amazon Q Developer operational investigations with a third-party ticketing system. Integrating with a ticketing tool enables Amazon Q Developer to send information about an investigation to that ticketing tool. For information about how to create this integration when you create an investigation group, see step 11b in Set up operational investigations.

This feature is available only in the US East (N. Virginia) Region.

Integration with third-party chat systems

By integrating Amazon Q Developer operational investigations with Amazon Q Developer in chat applications, you can have updates from investigations sent to third-party chat services, including Slack, and Microsoft Teams. The integration is facilitated by Amazon Simple Notification Service.

To integrate with Amazon Q Developer in chat applications, you must complete three steps. We recommend completing the steps in the following order.

- Create an Amazon SNS topic and add an access policy to it
- Configure in the Amazon Q Developer in chat applications console
- Configure in the CloudWatch console

Topics

Create and configure the Amazon SNS topic

Create an Amazon SNS topic in US East (N. Virginia) to use for the integration. For more information, see Creating an Amazon Simple Notification Service topic.

To enable Amazon Q Developer operational investigations to send notifications, you must add an the following access policy to the Amazon SNS topic

```
{
    "Sid": "AIOPS-CHAT-PUBLISH",
    "Effect": "Allow",
    "Principal": {
        "Service": "aiops.amazonaws.com"
    },
    "Action": "sns:Publish",
```

```
    "Resource": "SNS-TOPIC-ARN",
    "Condition": {
        "StringEquals": {
            "aws:SourceAccount": "account-Id"
        }
    }
}
```

Configure Amazon Q Developer in chat applications

To configure Amazon Q Developer in chat applications for communication with a third-party chat service, follow the instructions in one of the following links:

- Tutorial: Get started with Slack.
- Tutorial: Get started with Microsoft Teams.

Then, to support using AI assistant actions within chat channels you must provide the Amazon Q Developer in chat applications role with appropriate permissions. When you create a new IAM channel role for the channel, select the **Notifications** and **Amazon Q operations assistant permissions** policy templates.

Attach the **AIOpsOperatorAccess** managed IAM policy to the guardrail policies in Amazon Q Developer in chat applications. This grants permissions to Amazon Q Developer in chat applications to interact with Amazon Q Developer operational investigations and perform required actions on your behalf.

In the channel configuration, you must also subscribe to the Amazon SNS topic that you created in the previous step.

Amazon SNS

You must use the CloudWatch console to configure Amazon Q Developer operational investigations to integrate with Amazon SNS. You can do this while you create the investigation group in your account, or later.

For information about completing the step while you create the investigation group, see Step 9b at Set up operational investigations.

If you have already created an investigation group and want to add chat integration, follow these steps.

To add chat integration to an existing investigation group

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. Choose **AI operations**, **Configuration**.
3. Choose the **Third-party integrations** tab.
4. In the **Chat integration** section, do the following:

   - If you have already integrated Amazon Q Developer in chat applications with a third-party chat system, you can choose **Select SNS topic** to choose the Amazon SNS topic to use to send updates to about investigations. This Amazon SNS topic will relay those updates to the chat client.
   - If you want to integrate Amazon Q Developer in chat applications with a third-party chat system, choose **Configure new chat client**. For more information about setting up this configuration, see Getting started with Amazon Q Developer in chat applications.

Amazon IAM Identity Center

In the US East (N. Virginia) Region, you can integrate Amazon Q Developer operational investigations with Amazon IAM Identity Center. Before you create an Amazon Q Developer operational investigations application in IAM Identity Center, make sure you complete the following prerequisites:

- Enable an organization-level **IAM Identity Center instance** in your management account and **connect the identity source** in IAM Identity Center. Amazon Q Developer operational investigations doesn't support account-level IAM Identity Center instances.

  > **Note**
  >
  > To minimize latency, we recommend that you use an IAM Identity Center instance created in the same Region as your Amazon Q Developer operational investigations application. However, you can also use an IAM Identity Center instance created in an Amazon Region not yet supported by Amazon Q Developer operational investigations. For more information, see Cross-Region IAM Identity Center integration.

- Enable the **identity-aware session** on your IAM Identity Center instance.

Cross-Region IAM Identity Center integration

Amazon Q Developer operational investigations can integrate with IAM Identity Center in any commercial region where IAM Identity Center is available, including opt-in Regions. This integration works even if the Region isn't directly supported by Amazon Q Developer operational investigations. You have the flexibility to use an IAM Identity Center instance configured in a Rgion different from where Amazon Q Developer operational investigations is available.

When your IAM Identity Center instance is in a different Region than Amazon Q Developer operational investigations, you enable Amazon Q to make inter-Region calls to access information from your IAM Identity Center instance, such as user and application attributes. This capability allows Amazon Q Developer operational investigations to support IAM Identity Center-enabled applications regardless of regional differences. In this setup, your Amazon Q Developer operational investigations application will have access to user and application information from an IAM Identity Center instance deployed in another Region.

If your IAM Identity Center instance is in a different Region than your Amazon Q Developer operational investigations application, you might experience higher latency when using Amazon Q Developer operational investigations. This is caused by the increased overhead of making inter-Region calls. The increase in latency will be proportional to the distance between the two Regions.

(Recommended) Best practices to enhance investigations

As a best practice, we recommend that you enable several Amazon services and features in your account that can help Amazon Q Developer discover more information in your topology and make better suggestions during investigations.

> **Note**
>
> The Amazon Q Developer operational investigations feature is in preview release and is subject to change. It is currently available in the following Regions:
>
> - US East (N. Virginia)
> - US East (Ohio)
> - US West (Oregon)
> - Asia Pacific (Hong Kong)
> - Asia Pacific (Mumbai)

- Asia Pacific (Singapore)
- Asia Pacific (Sydney)
- Asia Pacific (Tokyo)
- Europe (Frankfurt)
- Europe (Ireland)
- Europe (Spain)
- Europe (Stockholm)

Topics

CloudWatch agent

We recommend that you install the latest version of the CloudWatch agent on your servers. Using a recent version of the CloudWatch agent enhances the ability to find issues in Amazon EC2 and Amazon EBS during investigations. At a minimum, you should use Version 1.300049.1 or later of the CloudWatch agent. For more information about the CloudWatch agent and how to install it, see Collect metrics, logs, and traces with the CloudWatch agent.

Amazon CloudTrail

We recommend that you enable CloudTrail including trails in your investigations. CloudTrail records events about changes in your system including deployment events. These events can often be useful to Amazon Q Developer to create hypotheses about root causes of issues in your system. For more information, see **What is Amazon CloudTrail** and **Working with CloudTrail trails**.

CloudWatch Application Signals

CloudWatch Application Signals discovers the topology of your environment, including your applications and their dependencies. It also automatically collects standard metrics such as latency and availability. By enabling Application Signals, Amazon Q Developer can use this topology and metric information during investigations.

For more information about application signals, see Application Signals.

X-Ray

We recommend that you enable Amazon X-Ray. X-Ray collects traces about requests that your applications serve. For any traced request to your application, you can see detailed information not only about the request and response, but also about calls that your application makes to downstream Amazon resources, microservices, databases, and web APIs. This information can help Amazon Q Developer during investigations.

For more information, see **What is Amazon X-Ray**

Container insights

If you use Amazon ECS or Amazon EKS, we recommend that you install Container insights. This improves the ability of Amazon Q Developer to find issues in your containers. For more information about the CloudWatch agent and how to install it, see Container Insights.

Operational investigation data retention

> **Note**
>
> The Amazon Q Developer operational investigations feature is in preview release and is subject to change. It is currently available in the following Regions:
>
> - US East (N. Virginia)
> - US East (Ohio)
> - US West (Oregon)
> - Asia Pacific (Hong Kong)
> - Asia Pacific (Mumbai)
> - Asia Pacific (Singapore)
> - Asia Pacific (Sydney)
> - Asia Pacific (Tokyo)
> - Europe (Frankfurt)

- Europe (Ireland)
- Europe (Spain)
- Europe (Stockholm)

The retention period that you set for an investigation group determines how long that investigation data is kept. Valid values are seven days to 90 days.

After you first create an investigation, if you don't end it manually, it moves to a CLOSED state automatically after seven days. Then, the retention period determines how long the data is kept after the investigation moves to the CLOSED state. The data that is kept during the retention period includes the data in the investigation, accepted and discarded findings, and AI assistant audit log messages.

When this retention period expires, the investigation data is deleted.

If you manually end an investigation, that also moves the investigation to the CLOSED state and the retention period time begins to be in effect.

Insights that Amazon Q Developer can surface in investigations

Note

The Amazon Q Developer operational investigations feature is in preview release and is subject to change. It is currently available in the following Regions:

- US East (N. Virginia)
- US East (Ohio)
- US West (Oregon)
- Asia Pacific (Hong Kong)
- Asia Pacific (Mumbai)
- Asia Pacific (Singapore)
- Asia Pacific (Sydney)
- Asia Pacific (Tokyo)
- Europe (Frankfurt)
- Europe (Ireland)

- Europe (Spain)
- Europe (Stockholm)

Amazon Q Developer can surface the following types of items and add them to the **Suggestions** tab of an investigation.

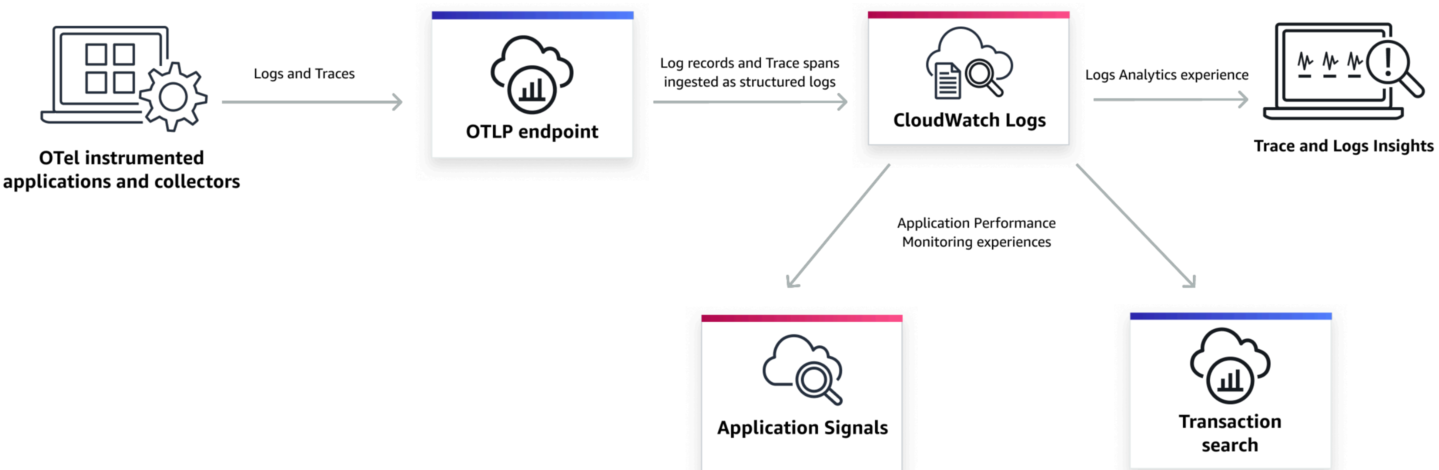
- Hypotheses about root causes
- CloudWatch alarms, including both metric alarms and composite alarms
- CloudWatch metrics
- Amazon Health events
- Change events logged in CloudTrail
- X-Ray trace data
- CloudWatch Logs Insights queries for log groups in the Standard log class
- CloudWatch Contributor Insights data
- CloudWatch Application Signals data

OpenTelemetry

OpenTelemetry is an open-source observability framework that provides IT teams with standardized protocols and tools for collecting and routing telemetry data. It delivers a unified format for instrumenting, generating, gathering, and exporting application telemetry data, such as metrics, logs, and traces to monitoring platforms for analysis and insights. By using OpenTelemetry, you can avoid vendor lock-in, ensuring flexibility in the observability solutions.

You can use OpenTelemetry to directly send logs and traces to an OpenTelemetry Protocol (OTLP) endpoint, and get out-of-the box features like Logs Insights, LiveTail, and application performance monitoring experiences in [CloudWatch Application Signals](#).

Application Signals provides you with a unified, application-centric view of your applications, services, and dependencies, and helps you monitor and triage application health. You can also explore OTLP spans using the interactive search and analytics experience in CloudWatch to answer any questions related to application performance or end-user impact with [Transaction Search](#). You can also detect the impact on end users, find transactions in context of those issues using relevant attributes such as customer name or order number, correlate transactions to business events such as failed payments, and dive into interactions between application components to establish a root cause. Using CloudWatch, you can get complete application transaction coverage with correlated insights, helping you to accelerate mean time to resolution.



Topics

- [OTLP Endpoints](#)
- [Getting started](#)
- [Troubleshooting](#)

OTLP Endpoints

OpenTelemetry Protocol (OTLP) is a general-purpose telemetry data delivery protocol designed for the OpenTelemetry. CloudWatch OpenTelemetry endpoints are HTTP 1.1 endpoints. You need to configure your OpenTelemetry collector to start sending open telemetry data to CloudWatch. For more information, see Getting started. The endpoint authenticates callers using Signature 4 authentication. For more information, see Amazon Signature Version 4 for API requests.

Traces endpoint

The traces endpoint follows the pattern `https://xray.`*`Amazon Region`*`.amazonaws.com/v1/traces`. For example, for the US West (Oregon) (us-west-2) Region, the endpoint will be `https://xray.us-west-2.amazonaws.com/v1/traces`.

You need to configure your OpenTelemetry collector to start sending traces to CloudWatch. To get started, see Getting started.

Logs endpoint

The logs endpoint follows the pattern `https://logs.`*`Amazon Web Services Region`*`.amazonaws.com/v1/logs`. For example, for the `US West (Oregon) (us-west-2) Region`, the endpoint will be `https://logs.us-west-2.amazonaws.com/v1/logs`. You can use the above endpoint to forward logs to an existing `LogGroup` and `LogStream`. For more information on setting up `LogGroup` to ingest log data, see Amazon CloudWatch Logs concepts. You must configure `LogGroup` and `LogStream` when you invoke CloudWatch Logs OpenTelemetry endpoint by setting `x-aws-log-group` and `x-aws-log-stream` HTTP headers to `LogGroup` and `LogStream` name respectively. For more information, see Getting started.

Endpoint limits and restrictions

The table lists the common endpoint limits and restrictions for traces and logs.

Limit	Endpoint	Additional informati on
Required collector extension	sigv4authextension	To send traces to OTLP endpoint you

Limit	Endpoint	Additional informati on
		must use sigv4auth extension
Supported protocol	HTTP	The endpoint supports only HTTP and doesn't support gRPC
Supported OTLP versions	OTLP 1.x	
Payload format	binary, json	The endpoint accepts requests using binary and json formats
Compression methods	gzip, none	The endpoint only supports gzip and none compression methods

The table lists the endpoint limits and restrictions for traces.

Limit	Traces endpoint	Additional informati on
Maximum uncompressed bytes / request	5 MB	The OTLP endpoint will reject requests larger than 5MB when payload is uncompressed.
Maximum events / request	10,000 spans	The max allowed number of spans in one request is 10,000. Exceeding this limit will cause

Limit	Traces endpoint	Additional information
		rejection of the API call.
Single resource and scope size	16 KB	Each unique resource and corresponding scope should not exceed 16 KB of size. Exceeding this limit for any resource will cause rejection of the entire API call.
Single span maximum size	200 KB	Spans more than 200KB will be rejected by the endpoint.
Span created timestamps	2 hours in the future and 14 days in the past	None of the spans in the batch can be more than two hours in the future or more than 14 days in the past.
Maximum time gap in events / request	24 hours	

The table lists the endpoint limits and restrictions for logs.

Limit	Logs endpoint	Additional information
Maximum uncompressed bytes / request	1 MB	The OTLP endpoint will reject requests larger than 1MB

Limit	Logs endpoint	Additional information
		when payload is uncompressed. The maximum request size is 1,048,576 bytes after decompression and deserialization of binary data serialized by Protocol buffers. This size is calculated as the sum of all event messages in UTF-8, plus 26 bytes for each log record.
Request per second	5000	5000 transactions per second per account per Region You can request an increase to the per-second throttling quota by using the Service Quotas service.
Maximum events / request	10,000 logs	The max allowed number of spans in one request is 10,000. Exceeding this limit will cause rejection of the API call.

Limit	Logs endpoint	Additional information
Single resource and scope size	16 KB	Each unique resource and corresponding scope should not exceed 16 KB of size. Exceeding this limit for any resource will cause rejection of the entire API call.
Single LogEvent size	1 MB	LogEvent size is calculated as sum of sizes for each LogRecord, Scope and Resource. This quota can't be changed.
Logs created timestamps	2 hours in future and 14 days old	The log records in the batch does not have to be in a chronological order. However, the log records in the batch cannot be more than 2 hours in the future and cannot be more than 14 days in the past. Also, none of the log records can be earlier than the retention period of the log group.
Maximum time gap in events / request	24 hours	

Note

The account limits for Logs are shared across the SDK and the new Logs endpoint.

Getting started

To get started with OpenTelemetry in CloudWatch, you can use the pre-packaged OpenTelemetry setup that is available with the CloudWatch agent along with the Amazon Distro for OpenTelemetry SDKs. This gives you the most integrated monitoring experience in CloudWatch.

Note

Make sure Transaction Search is enabled before you use the OTLP Endpoint.

Alternatively, you have the flexibility to use the OpenTelemetry Collector or your own custom OpenTelemetry Collector to directly send telemetry to the OTLP endpoint. You can use the Amazon Distro for OpenTelemetry to go collector-less and to send telemetry directly to the OTLP endpoint. Make an informed choice based on the feature support:

Feature	OpenTelemetry Collector Contrib	Custom OpenTelemetry Collector	Amazon Distro for OpenTelemetry
CloudWatch application signals (Application performance metrics, service discovery, and service map)	Yes	Yes	Yes
Search and analyze spans and trace summaries	Yes	Yes	Yes
Search and analyze logs summaries	Yes	Yes	Yes

Feature	OpenTelemetry Collector Contrib	Custom OpenTelemetry Collector	Amazon Distro for OpenTelemetry
Application performance monitoring telemetry enrichment with Amazon infrastructure attributes that your application is hosted in.	No	Yes	Yes
Runtime metrics correlated with your application. For example, JVM metrics	No	Yes	No
Amazon Support	Data received by Amazon	Data received by Amazon	Data received by Amazon
Telemetry supported	Logs, Traces	Logs, Traces, Metrics	Traces

Topics

- OpenTelemetry Collector Contrib
- Build your own custom OpenTelemetry Collector
- Amazon Distro for OpenTelemetry

OpenTelemetry Collector Contrib

You can use the OpenTelemetry Collector Contrib to get started with OpenTelemetry in CloudWatch.

Prerequisite

Make sure *Transaction Search* is enabled in CloudWatch. For more information, see Transaction Search.

Download the OpenTelemetry Collector Contrib

Download the latest release of the OpenTelemetry Collector Contrib distribution.

Install the OpenTelemetry Collector Contrib

Install the OpenTelemetry Collector Contrib on any operating system and platform. For more information, see Install the Collector.

Setup Amazon credentials on your Amazon EC2 or on-premise hosts

You can setup Amazon credentials on your Amazon EC2 or on-premise hosts.

Setup IAM permissions for Amazon EC2

Follow the below procedure to attach the `CloudWatchAgentServerPolicy` IAM policy to the IAM role of your Amazon EC2 instance.

1. Open the IAM console at https://console.amazonaws.cn/iam/.
2. Choose **Roles** and find and select the role used by your Amazon EC2 instance.
3. Under the **Permissions** tab, choose **Add permissions**, **Attach policies**.
4. Using the search box, search for `CloudWatchAgentServerPolicy` policy.
5. Select the **CloudWatchAgentServerPolicy** policy and choose **Add permissions**.

Setup IAM permissions for on-premise hosts

You can create an IAM user that can be used to provide permissions to your on-premise hosts.

1. Open the IAM console at https://console.amazonaws.cn/iam/.
2. Choose **Users**, **Create User**.
3. Under **User details**,for **User name**, enter a name for the new IAM user. This is the sign-in name for Amazon that will be used to authenticate your host.
4. Choose **Next**.
5. On the **Set permissions** page, under **Permissions options**, select **Attach policies directly**.
6. From the **Permissions policies** list, select the **CloudWatchAgentServerPolicy** policy to add to your user.

7. Choose **Next**.
8. On the **Review and create** page, ensure that you are satisfied with the user name and that the **CloudWatchAgentServerPolicy** policy is under the **Permissions summary**.
9. Choose **Create user**.
10. **Create and retrieve your Amazon access key and secret key** – In the navigation pane in the IAM console, choose **Users** and then select the user name of the user that you created in the previous step.
11. On the user's page, choose the **Security credentials** tab.
12. Under the **Access keys** section, choose **Create access key**.
13. For **Create access key Step 1**, choose **Command Line Interface (CLI)**.
14. For **Create access key Step 2**, optionally enter a tag and then choose **Next**.
15. For **Create access key Step 3,** select **Download .csv file** to save a .csv file with your IAM user's access key and secret access key. You need this information for the next steps.
16. Choose **Done**.
17. Configure your Amazon credentials in your on-premises host by entering the following command. Replace *ACCESS_KEY_ID* and *SECRET_ACCESS_ID* with your newly generated access key and secret access key from the .csv file that you downloaded in the previous step.

```
$ aws configure
AWS Access Key ID [None]: ACCESS_KEY_ID
AWS Secret Access Key [None]: SECRET_ACCESS_ID
Default region name [None]: MY_REGION
Default output format [None]: json
```

Setup Amazon credentials for your Amazon EKS or Kubernetes clusters

To setup Amazon credentials for your Amazon EKS or Kubernetes clusters to send telemetry to CloudWatch, follow the below procedure.

Setup IAM permissions for Amazon EKS

1. Create an IAM OIDC identity provider for your cluster using the following command.

```
eksctl utils associate-iam-oidc-provider --cluster ${CLUSTER_NAME} --region
 ${REGION} --approve
```

2. Assign IAM roles to Kubernetes service account for OTel Collector using the following command.

```
eksctl create iamserviceaccount \
--name ${COLLECTOR_SERVICE_ACCOUNT}\
--namespace ${NAMESPACE} \
--cluster ${CLUSTER_NAME} \
--region ${REGION} \
--attach-policy-arn arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy \
--approve \
--override-existing-serviceaccounts
```

Setup IAM permissions for Kubernetes

1. Configure your Amazon credentials in your on-premises host by entering the following command. Replace *ACCESS_KEY_ID* and *SECRET_ACCESS_ID* with your newly generated access key and secret access key from the .csv file that you downloaded in the previous step. By default, the credential file is saved under */home/user/.aws/credentials.*.

```
aws configure
AWS Access Key ID [None]: ACCESS_KEY_ID
AWS Secret Access Key [None]: SECRET_ACCESS_ID
Default region name [None]: MY_REGION
Default output format [None]: json
```

2. Edit OpenTelemetry Collector resource to add the newly created AWS credentials secret by using the command: `kubectl edit OpenTelemetryCollector otel_collector`
3. Using the file editor, add the Amazon credentials into the OpenTelemetryCollector container by adding the following configuration to the top of the deployment. Replace the path `/home/user/.aws/credentials` with the location of your local Amazon credentials file.

```
spec:
volumeMounts:
- mountPath: /rootfs
volumeMounts:
- name: aws-credentials
mountPath: /root/.aws
readOnly: true
volumes:
- hostPath:
path: /home/user/.aws/credentials
name: aws-credentials
```

Configure the OpenTelemetry Collector

Copy and paste the content below to configure your collector to send logs and traces to the OTLP endpoints.

```
receivers:
  otlp:
    protocols:
      grpc:
        endpoint: 0.0.0.0:4317
      http:
        endpoint: 0.0.0.0:4318

exporters:
  otlphttp/logs:
    compression: gzip
    logs_endpoint: logs_otlp_endpoint
    headers:
      x-aws-log-group: ency_log_group
      x-aws-log-stream: default
    auth:
      authenticator: sigv4auth/logs

  otlphttp/traces:
    compression: gzip
    traces_endpoint: traces_otlp_endpoint
    auth:
      authenticator: sigv4auth/traces
```

```
extensions:
  sigv4auth/logs:
    region: "region"
    service: "logs"
  sigv4auth/traces:
    region: "region"
    service: "xray"

service:
  telemetry:
  extensions: [sigv4auth/logs, sigv4auth/traces]
  pipelines:
    logs:
      receivers: [otlp]
      exporters: [otlphttp/logs]
    traces:
      receivers: [otlp]
      exporters: [otlphttp/traces]
```

The following is an example to send logs and traces using sigv4 to us-east-1.

```
receivers:
  otlp:
    protocols:
      grpc:
        endpoint: 0.0.0.0:4317
      http:
        endpoint: 0.0.0.0:4318

exporters:
  otlphttp/logs:
    compression: gzip
    logs_endpoint: https://logs.us-east-1.amazonaws.com/v1/logs
    headers:
      x-aws-log-group: MyApplicationLogs
      x-aws-log-stream: default
    auth:
      authenticator: sigv4auth/logs

  otlphttp/traces:
    compression: gzip
```

```
    traces_endpoint: https://xray.us-east-1.amazonaws.com/v1/traces
    auth:
      authenticator: sigv4auth/traces

extensions:
  sigv4auth/logs:
    region: "us-east-1"
    service: "logs"
  sigv4auth/traces:
    region: "us-east-1"
    service: "xray"

service:
  telemetry:
  extensions: [sigv4auth/logs, sigv4auth/traces]
  pipelines:
    logs:
     receivers: [otlp]
      exporters: [otlphttp/logs]
    traces:
      receivers: [otlp]
      exporters: [otlphttp/traces]
```

Note

Configure your OpenTelemetry SDKs to *always_on* sampling configuration to reliably record 100% spans and get full visibility into your critical applications with CloudWatch Application Signals. For more information, see an OpenTelemetry Java SDK sampler configuration example. For an example on setting up OpenTelemetry Collector with X-Ray OTLP endpoint, see the application signals demo repository.

Build your own custom OpenTelemetry Collector

You can build your own custom OpenTelemetry Collector to get the best application observability experience in CloudWatch with OpenTelemetry. In this setup, you need to build your own OpenTelemetry Collector with open source CloudWatch components.

Prerequisite

Make sure *Transaction Search* is enabled in CloudWatch. For more information, see Transaction Search.

Build your own collector

You can build your own collector with the following configuration to monitor your application in CloudWatch with OpenTelemetry. For more information, see Building a custom collector.

The common configuration for CloudWatch.

```
dist:
  name: otelcol-dev
  description: OTel Collector for sending telemetry to CloudWatch.
  output_path: ./otelcol-dev
extensions:
  - gomod: github.com/open-telemetry/opentelemetry-collector-contrib/extension/
sigv4authextension v0.111.0
  - gomod: github.com/open-telemetry/opentelemetry-collector-contrib/extension/awsproxy
 v0.113.0
exporters:
  - gomod: go.opentelemetry.io/collector/exporter/otlpexporter v0.111.0
  - gomod: go.opentelemetry.io/collector/exporter/otlphttpexporter v0.111.0
receivers:
  - gomod: go.opentelemetry.io/collector/receiver/otlpreceiver v0.111.0
```

Additional configuration for traces.

```
# Enable Tracing
dist:
  name: otelcol-dev
  description: OTel Collector for sending telemetry to CloudWatch.
  output_path: ./otelcol-dev
extensions:
    #Include common configurations and your custom extensions

exporters:
    #Include common configurations and your custom extensions
```

```
receivers:
  - gomod: go.opentelemetry.io/collector/receiver/otlpreceiver v0.111.0
processors:
  - gomod: github.com/amazon-contributing/opentelemetry-collector-contrib/processor/
awsapplicationsignalsprocessor v0.113.0
  - gomod: github.com/open-telemetry/opentelemetry-collector-contrib/processor/
resourcedetectionprocessor v0.113.0
  - gomod: github.com/open-telemetry/opentelemetry-collector-contrib/processor/
metricstransformprocessor v0.113.0
replaces:
  - github.com/open-telemetry/opentelemetry-collector-contrib/internal/aws/awsutil
 v0.113.0 => github.com/amazon-contributing/opentelemetry-collector-contrib/internal/
aws/awsutil v0.113.0
  - github.com/open-telemetry/opentelemetry-collector-contrib/internal/aws/cwlogs
 v0.113.0 => github.com/amazon-contributing/opentelemetry-collector-contrib/internal/
aws/cwlogs v0.113.0
  - github.com/open-telemetry/opentelemetry-collector-contrib/exporter/awsemfexporter
 v0.113.0 => github.com/amazon-contributing/opentelemetry-collector-contrib/exporter/
awsemfexporter v0.113.0
  - github.com/openshift/api v3.9.0+incompatible => github.com/openshift/api
 v0.0.0-20180801171038-322a19404e37
```

Note

Note the following:

- After the collector is built, deploy and configure the custom collector in a host or kubernetes environment by following the procedure under OpenTelemetry Collector Contrib.
- For information on setting up custom OpenTelemetry collector with Application Signals Processor, see an Application Signals custom configurationexample. Application Signals Processor only supports the latest versions of the OpenTelemetry Collectors for custom builds. For information on the supported versions, see opentelemetry-collector-contrib repository.

Amazon Distro for OpenTelemetry

You can use the Amazon Distro for OpenTelemetry to go collector-less and to send traces directly to the OTLP endpoint (for traces).

> **Note**
>
> By default, Application Signals is enabled when you enable Transaction Search. Application Signals is not supported on Amazon Distro for OpenTelemetry and must be disabled.

Topics

Prerequisite

Make sure Transaction Search is enabled to send spans to the X-Ray OTLP endpoint. For more information, see Getting started with Transaction Search.

Set up IAM permissions for Amazon EC2

Follow these steps to attach the `AWSXrayWriteOnlyPolicy` IAM policy to the IAM role of your Amazon EC2 instance:

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. Choose **Roles** and find and select the role used by your Amazon EC2 instance.
3. Under the **Permissions** tab, choose **Add permissions**, then **Attach policies**.
4. Using the search box, search for the `AWSXrayWriteOnlyPolicy` policy.
5. Select the `AWSXrayWriteOnlyPolicy` policy and choose **Add permissions**.

Set up IAM permissions for on-premise hosts

Follow these steps to create an IAM user that can be used to provide permissions to your on-premise hosts.

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. Choose **Users** and then **Create User**.

3. Choose **Users**, **Create User**.
4. Under **User details**, for **User name**, enter a name for the new IAM user. This is the sign-in name for Amazon that will be used to authenticate your host.
5. Choose **Next**.
6. On the **Set permissions** page, under **Permissions options**, select **Attach policies directly**.
7. From the **Permissions policies** list, select the `AWSXrayWriteOnlyPolicy` policy to add to your user.
8. Choose **Next**.
9. On the **Review and create** page, make sure that you are satisfied with the user name and that the `AWSXrayWriteOnlyPolicy` policy is under the **Permissions summary**.
10. Choose **Create user**.
11. Create and retrieve your Amazon access key and secret key:

    1. In the navigation pane in the IAM console, choose **Users** and then select the user name of the user that you created in the previous step.
    2. On the user's page, choose the **Security credentials** tab.
    3. Under the **Access keys** section, choose **Create access key**.
    4. For **Create access key Step 1**, choose **Command Line Interface (CLI)**.
    5. For **Create access key Step 2**, optionally enter a tag and then choose **Next**.
    6. For **Create access key Step 3**, select **Download .csv file** to save a .csv file with your IAM user's access key and secret access key. You need this information for the next steps.
    7. Choose **Done**.
12. Configure your Amazon credentials in your on-premises host by entering the following command. Replace **`ACCESS_KEY_ID`** and **`SECRET_ACCESS_ID`** with your newly generated access key and secret access key from the .csv file that you downloaded in the previous step.

```
$ aws configure
AWS Access Key ID [None]: ACCESS_KEY_ID
AWS Secret Access Key [None]: SECRET_ACCESS_ID
Default region name [None]: MY_REGION
Default output format [None]: json
```

Enabling Amazon Distro for OpenTelemetry

You can enable traces for your application to be sent directly to the OTLP endpoint from Amazon Distro for OpenTelemetry on Java, Node.js, Python, and .Net.

Java

1. Download the latest version of the Amazon Distro for OpenTelemetry Java auto-instrumentation agent. You can download the latest version by using this command:

   ```
   curl -L -O https://github.com/aws-observability/aws-otel-java-instrumentation/
   releases/latest/download/aws-opentelemetry-agent.jar
   ```

   To view all the released versions, see aws-otel-java-instrumentation releases.
2. To enable the exporter that directly sends traces to the X-Ray OTLP traces endpoint and to optimize benefits, use the environment variables to provide additional information before you start your application.
3. For the `OTEL_RESOURCE_ATTRIBUTES` variable, specify the following information as key-value pairs:

   (Optional) `service.name` sets the name of the service. This will be displayed as the service name for your application in Application Signals dashboards. When you don't provide a value for this key, the default of `UnknownService` is used.

   (Optional) `deployment.environment` sets the environment that the application runs in. This will be diplayed as the **Hosted In** environment of your application. When you don't specify this, one of the following defaults is used:

   - If this is an instance that is part of an Auto Scaling group, it is set to `ec2:name-of-Auto-Scaling-group`
   - If this is an Amazon EC2 instance that is not part of an Auto Scaling group, it is set to `ec2:default`
   - If this is an on-premises host, it is set to `generic:default`

   This environment variable is used only by Application Signals, and is converted into X-Ray trace annotations and CloudWatch metric dimensions

4. For the OTEL_EXPORTER_OTLP_TRACES_ENDPOINT variable, specify the X-Ray OTLP traces endpoint: https://xray.[*AWSRegion*].amazonaws.com/v1/traces. For example:

```
export OTEL_EXPORTER_OTLP_TRACES_ENDPOINT="https://xray.us-west-2.amazonaws.com/
v1/traces"
```

5. For the JAVA_TOOL_OPTIONS variable, specify the path where the Amazon Distro for OpenTelemetry Java auto-instrumentation agent is stored.

```
export JAVA_TOOL_OPTIONS=" -javaagent:$AWS_ADOT_JAVA_INSTRUMENTATION_PATH"
```

   For example:

```
export AWS_ADOT_JAVA_INSTRUMENTATION_PATH="./aws-opentelemetry-agent.jar"
```

6. For the OTEL_METRICS_EXPORTER variable, it is recommended to set the value to none.
7. For the OTEL_LOGS_EXPORTER variable, it is recommended to set the value to none.
8. For the OTEL_TRACES_EXPORTER variable, you have to set the value for otlp (this is optional and is the default value if this environment variable is not set).
9. For the OTEL_EXPORTER_OTLP_PROTOCOL variable, you have to set the value to http/protobuf (this is optional and is the default value if this environment variable is not set). The X-Ray OTLP endpoint currently only supports the HTTP protocol.
10. Your application should now be running with Amazon Distro for OpenTelemetry Java instrumentation and will generate spans. These spans are stored in the aws/spans CloudWatch LogsLogGroup in your account. You can also view the traces and metrics correlated with your spans in the CloudWatch Traces and Metrics Console.
11. Start your application with the environment variables you set. Here is an example of a starting script.

   (Note: The following configuration supports only versions 1.32.2 and later of the Amazon Distro for OpenTelemetry auto-instrumentation agent for Java.)

```
JAVA_TOOL_OPTIONS=" -javaagent:$AWS_ADOT_JAVA_INSTRUMENTATION_PATH" \
OTEL_METRICS_EXPORTER=none \
OTEL_LOGS_EXPORTER=none \
OTEL_EXPORTER_OTLP_PROTOCOL=http/protobuf \
```

```
OTEL_EXPORTER_OTLP_TRACES_ENDPOINT=https://xray.us-east-1.amazonaws.com/v1/
traces \
OTEL_RESOURCE_ATTRIBUTES="service.name=$YOUR_SVC_NAME" \
java -jar $MY_JAVA_APP.jar
```

Node.js

1. Download the latest version of the Amazon Distro for OpenTelemetry JavaScript auto-instrumentation agent for Node.js. You can install using the command:

```
npm install @aws/aws-distro-opentelemetry-node-autoinstrumentation
```

   To view information about all released versions, see [Amazon Distro for OpenTelemetry JavaScript instrumentation]().

2. To enable the exporter that directly sends traces to the X-Ray OTLP endpoint and to optimize benefits, use the environment variables to provide additional information before you start your application.

3. For the `OTEL_RESOURCE_ATTRIBUTES` variable, specify the following information as key-value pairs:

   (Optional) `service.name` sets the name of the service. This will be displayed as the service name for your application in Application Signals dashboards. When you don't provide a value for this key, the default of `UnknownService` is used.

   (Optional) `deployment.environment` sets the environment that the application runs in. This will be diplayed as the **Hosted In** environment of your application in Application Signals dashboards. When you don't specify this variable, one of the following defaults is used:

   - If this is an instance that is part of an Auto Scaling group, it is set to `ec2:name-of-Auto-Scaling-group`
   - If this is an Amazon EC2 instance that is not part of an Auto Scaling group, it is set to `ec2:default`
   - If this is an on-premises host, it is set to `generic:default`

This environment variable is used only by Application Signals, and is converted into X-Ray trace annotations and CloudWatch metric dimensions.

4. For the `OTEL_EXPORTER_OTLP_TRACES_ENDPOINT` variable, specify the X-Ray OTLP traces endpoint: `https://xray.[AWSRegion].amazonaws.com/v1/traces`.

   For example:

   ```
   export OTEL_EXPORTER_OTLP_TRACES_ENDPOINT="https://xray.us-west-2.amazonaws.com/
   v1/traces"
   ```

5. For the `OTEL_METRICS_EXPORTER` variable, it is recommended to set the value to none. Application Signals metrics are generated by the OTLP endpoint.
6. For the `OTEL_LOGS_EXPORTER` variable, it is recommended to set the value to none.
7. For the `OTEL_TRACES_EXPORTER` variable, you have to set the value for `otlp` (this is optional and is the default value if this environment variable is not set).
8. For the `OTEL_EXPORTER_OTLP_PROTOCOL` variable, you have to set the value to `http/protobuf` (this is optional and is the default value if this environment variable is not set). The X-Ray OTLP endpoint currently only supports the HTTP protocol.
9. Your application should now be running with Amazon Distro for OpenTelemetry Java instrumentation and will generate spans. These spans are stored in the `aws/spans` CloudWatch LogsLogGroup in your account. You can also view the traces and metrics correlated with your spans in the CloudWatch Traces and Metrics Console.
10. Start your application with the environment variables you set. Here is an example of a starting script.

    (Note: Replace $*SVC_NAME* with your application name. This is displayed as the name of the application.

    ```
    OTEL_METRICS_EXPORTER=none \
    OTEL_LOGS_EXPORTER=none \
    OTEL_EXPORTER_OTLP_PROTOCOL=http/protobuf \
    OTEL_EXPORTER_OTLP_TRACES_ENDPOINT=https://xray.us-east-1.amazonaws.com/v1/
    traces \
    OTEL_RESOURCE_ATTRIBUTES="service.name=$SVC_NAME" \
    node —require '@aws/aws-distro-opentelemetry-node-autoinstrumentation/register'
     your-application.js
    ```

Python

1. Download the latest version of the Amazon Distro for OpenTelemetry Python auto-instrumentation agent. You can install using the command:

```
pip install aws-opentelemetry-distro
```

2. To enable the exporter to directly send traces to the X-Ray OTLP endpoint and to optimize benefits, use the environment variables to provide additional information before you start your application.

3. For the `OTEL_RESOURCE_ATTRIBUTES` variable, specify the following information as key-value pairs:

   (Optional) `service.name` sets the name of the service. This will be displayed as the service name for your application in Application Signals dashboards. When you don't provide a value for this key, the default of `UnknownService` is used.

   (Optional) `deployment.environment` sets the environment that the application runs in. This will be diplayed as the **Hosted In** environment of your application in Application Signals dashboards. When you don't specify this, one of the following defaults is used:

   - If this is an instance that is part of an Auto Scaling group, it is set to `ec2:name-of-Auto-Scaling-group`
   - If this is an Amazon EC2 instance that is not part of an Auto Scaling group, it is set to `ec2:default`
   - If this is an on-premises host, it is set to `generic:default`

   This environment variable is used only by Application Signals, and is converted into X-Ray trace annotations and CloudWatch metric dimensions.

4. For the `OTEL_EXPORTER_OTLP_TRACES_ENDPOINT` variable, specify the X-Ray OTLP traces endpoint: `https://xray.[`*`AWSRegion`*`].amazonaws.com/v1/traces`.

   For example:

```
export OTEL_EXPORTER_OTLP_TRACES_ENDPOINT="https://xray.us-west-2.amazonaws.com/
v1/traces"
```

5. For the OTEL_METRICS_EXPORTER variable, it is recommended to set the value to none. Application Signals metrics are generated by the OTLP endpoint.
6. For the OTEL_LOGS_EXPORTER variable, it is recommended to set the value to none.
7. For the OTEL_TRACES_EXPORTER variable, you have to set the value for otlp (this is optional and is the default value if this environment variable is not set).
8. For the OTEL_EXPORTER_OTLP_PROTOCOL variable, you have to set the value to http/protobuf (this is optional and is the default value if this environment variable is not set). The X-Ray OTLP endpoint currently only supports the HTTP protocol.
9. Your application should now be running with Amazon Distro for OpenTelemetry Java instrumentation and will generate spans. These spans are stored in the aws/spans CloudWatch LogsLogGroup in your account. You can also view the traces and metrics correlated with your spans in the CloudWatch Traces and Metrics Console.
10. Start your application with the environment variables you set. Here is an example of a starting script.

    (Note: Replace $*SVC_NAME* with your application name and replace $*PYTHON_APP* with the location and name of your application.

```
OTEL_METRICS_EXPORTER=none \
OTEL_LOGS_EXPORTER=none \
OTEL_PYTHON_DISTRO=aws_distro \
 OTEL_PYTHON_CONFIGURATOR=aws_configurator \
OTEL_EXPORTER_OTLP_PROTOCOL=http/protobuf \
OTEL_EXPORTER_OTLP_TRACES_ENDPOINT=https://xray.us-east-1.amazonaws.com/v1/
traces \
OTEL_RESOURCE_ATTRIBUTES="service.name=$SVC_NAME" \
opentelemetry-instrument python $MY_PYTHON_APP.py
```

.Net

To enable the exporter that directly sends traces to the X-Ray OTLP traces endpoint and to optimize benefits, set the environment variables to provide additional information before you start your application. These variables are also necessary to set up the .NET instrumentation.

1. Replace dotnet-service-name in the OTEL_RESOURCE_ATTRIBUTES environment variable with the service name of your choice.
2. Set OTEL_TRACES_EXPORTER=none.

3. Set OTEL_AWS_SIG_V4_ENABLED=true.

   An example for Linux.

```
export INSTALL_DIR=OpenTelemetryDistribution
export CORECLR_ENABLE_PROFILING=1
export CORECLR_PROFILER={918728DD-259F-4A6A-AC2B-B85E1B658318}
export CORECLR_PROFILER_PATH=${INSTALL_DIR}/linux-x64/
OpenTelemetry.AutoInstrumentation.Native.so
export DOTNET_ADDITIONAL_DEPS=${INSTALL_DIR}/AdditionalDeps
export DOTNET_SHARED_STORE=${INSTALL_DIR}/store
export DOTNET_STARTUP_HOOKS=${INSTALL_DIR}/net/
OpenTelemetry.AutoInstrumentation.StartupHook.dll
export OTEL_DOTNET_AUTO_HOME=${INSTALL_DIR}

export
 OTEL_DOTNET_AUTO_PLUGINS="AWS.Distro.OpenTelemetry.AutoInstrumentation.Plugin,
 AWS.Distro.OpenTelemetry.AutoInstrumentation"
export OTEL_TRACES_EXPORTER=none
export OTEL_AWS_SIG_V4_ENABLED=true

export OTEL_RESOURCE_ATTRIBUTES=service.name=dotnet-service-name
export OTEL_METRICS_EXPORTER=none
export OTEL_LOGS_EXPORTER=none
export OTEL_EXPORTER_OTLP_PROTOCOL=http/protobuf
export OTEL_EXPORTER_OTLP_TRACES_ENDPOINT=https://xray.us-east-1.amazonaws.com/
v1/traces
```

   An example for Windows Server.

```
$env:INSTALL_DIR = "OpenTelemetryDistribution"
$env:CORECLR_ENABLE_PROFILING = 1
$env:CORECLR_PROFILER = "{918728DD-259F-4A6A-AC2B-B85E1B658318}"
$env:CORECLR_PROFILER_PATH = Join-Path $env:INSTALL_DIR "win-x64/
OpenTelemetry.AutoInstrumentation.Native.dll"
$env:DOTNET_ADDITIONAL_DEPS = Join-Path $env:INSTALL_DIR "AdditionalDeps"
$env:DOTNET_SHARED_STORE = Join-Path $env:INSTALL_DIR "store"
$env:DOTNET_STARTUP_HOOKS = Join-Path $env:INSTALL_DIR "net/
OpenTelemetry.AutoInstrumentation.StartupHook.dll"
$env:OTEL_DOTNET_AUTO_HOME = $env:INSTALL_DIR
$env:OTEL_DOTNET_AUTO_PLUGINS =
 "AWS.Distro.OpenTelemetry.AutoInstrumentation.Plugin,
 AWS.Distro.OpenTelemetry.AutoInstrumentation"
```

```
$env:OTEL_TRACES_EXPORTER=none
$env:OTEL_AWS_SIG_V4_ENABLED=true

$env:OTEL_RESOURCE_ATTRIBUTES=service.name=dotnet-service-name
$env:OTEL_METRICS_EXPORTER=none
$env:OTEL_LOGS_EXPORTER=none
$env:OTEL_EXPORTER_OTLP_PROTOCOL=http/protobuf
$env:OTEL_EXPORTER_OTLP_TRACES_ENDPOINT=https://xray.us-east-1.amazonaws.com/v1/
traces
```

4. Start your application with the environment variables.
5. (Optional) Alternatively, you can use the installation scripts provided to help installation and setup of Amazon Distro for OpenTelemetry .NET auto-instrumentation package.

   For Linux, download and install the Bash installation script from the GitHub releases page:

```
# Download and Install
curl -L -O https://github.com/aws-observability/aws-otel-dotnet-instrumentation/
releases/latest/download/aws-otel-dotnet-install.sh
chmod +x ./aws-otel-dotnet-install.sh
./aws-otel-dotnet-install.sh
# Instrument
. $HOME/.otel-dotnet-auto/instrument.shexport
 OTEL_RESOURCE_ATTRIBUTES=service.name=dotnet-service-name
```

   For Windows Server, download and install the PowerShell installation script from the GitHub releases page:

```
# Download and Install
$module_url = "https://github.com/aws-observability/aws-otel-dotnet-
instrumentation/releases/latest/download/AWS.Otel.DotNet.Auto.psm1"
$download_path = Join-Path $env:temp "AWS.Otel.DotNet.Auto.psm1"
Invoke-WebRequest -Uri $module_url -OutFile $download_path
Import-Module $download_path
Install-OpenTelemetryCore
# Instrument
Import-Module $download_path
Register-OpenTelemetryForCurrentSession -OTelServiceName "dotnet-service-name"
Register-OpenTelemetryForIIS
```

You can find the NuGet package of the Amazon Distro for OpenTelemetry .NET auto-instrumentation package in the official NuGet repository. Make sure to check the README file for instructions.

Troubleshooting

The following are the common troubleshooting scenarios and solutions for OTLP endpoint.

Issue	Description	Solution
Non-existing Amazon credentials when launching OCB collector	Collector throws the following error when starting. *Error: invalid configura tion: extensions::sigv4auth: could not retrieve credentia l provider: failed to refresh cached credentials, no EC2 IMDS role found, operation error ec2imds: GetMetada ta, request canceled, context deadline exceeded*.	Enter the correct credentials.
Invalid Amazon credentials	Collector throws HTTP Status Code 403, Message=The security token included in the request is invalid., Details=[ ]“ when sending requests though OTLP endpoint.	Refresh the Amazon credentia ls on the collector server.
Transactions Search disabled	Collector throws Message=T he OTLP API is supported with CloudWatch Logs as a Trace Segment Destination.	Make sure Transaction Search is enabled in CloudWatc h before using the OTLP endpoint for traces. For more information, see Transaction Search.

Issue	Description	Solution
Batching and timeout issues	Collector throws one of these issues: • max elapsed time expired failed to make an HTTP request • io.opentelemetry.exporter.internal.http.HttpExporter - Failed to export spans. The request could not be executed. Full error message: timeout • io.opentelemetry.exporter.internal.grpc.GrpcExporter - Failed to export spans. Server responded with gRPC status code 2. Error message: timeout • rpc error: code = DeadlineExceeded desc = context deadline exceeded • rpc error: code = ResourceExhausted desc = Too many requests", "dropped_items": 1024	Tune batching and timeout policies using batchprocessor.

Issue	Description	Solution
Retry issues	Transient network issues between the collector and OTLP endpoint. • rpc error: code = Unavailab le desc = error reading from server: read tcp • rpc error: code = Unavailab le desc = unexpected HTTP status code received from server: 502 (Bad Gateway); • rpc error: code = Unavailab le desc = unexpected HTTP status code received from server: 503 (Service Unavailable)	Tune retry policy using exporter.
Payload rejected	NA	Make sure the payload sent to the trace endpoint is within the limits and restrictions. For more information, see Endpoint limits and restricti ons.

Network Monitoring

The topics in this section describe CloudWatch network and internet monitoring capabilities provided by Network Flow Monitor, Internet Monitor and Network Synthetic Monitor. These services help you to gain operational visibility into the network and internet performance and availability of your applications hosted on Amazon.

- Network Flow Monitor provides near real-time visibility into network performance, such as packet loss and latency, for traffic between Amazon EC2 instances, as well as traffic toward other Amazon services, such as Amazon S3 and Amazon DynamoDB. Network Flow Monitor works by using data from lightweight software agents that you install to run on your instances. These fully-managed agents gather performance statistics from TCP connections and send them to the Network Flow Monitor backend. By creating monitors for specific agents and then using Network Flow Monitor dashboards, you can quickly visualize packet loss and latency of your network connections, and use attribution information to determine where to focus your troubleshooting efforts to improve your end users' experience.
- Internet Monitor uses the connectivity data that Amazon captures from its global networking footprint to calculate a baseline of performance and availability for internet-facing traffic. You can see a global view of traffic patterns and health events, and easily drill down into information about events. You can also get alerts for internet health events that affect your application clients. In addition, you can use insights that Internet Monitor provides to explore potential improvements to your client experience, by using Amazon CloudFront or routing through different Amazon Web Services Regions.
- Network Synthetic Monitor uses fully-managed agents to enable you to track and visualize latency and packet loss for hybrid network connections. To gather measurements and enable Network Synthetic Monitor to create health event alerts for your application, you create probes that are sent from your resources hosted on Amazon to on-premises destination IP addresses. You don't need to install additional agents to monitor your network performance. As with Internet Monitor, you can set alerts and thresholds, get information to help you quickly troubleshoot issues, and then take action to improve your end user experience.

Topics

- Using Network Flow Monitor
- Using Internet Monitor
- Using Network Synthetic Monitor

Using Network Flow Monitor

Network Flow Monitor provides near real-time visibility into network performance, such as packet loss and latency, for traffic between Amazon EC2 instances, as well as traffic toward other Amazon services, such as Amazon S3 and Amazon DynamoDB. Network Flow Monitor receives data from lightweight software agents that you install on your instances. The agents gather performance statistics from TCP connections. This data is sent to the Network Flow Monitor backend service, and the top contributors for each metric type are calculated. Network Flow Monitor also determines if Amazon is the cause of a detected network issue, and reports that information for network flows that you choose to monitor details for.

You can view network performance information for network flows for resources in a single account, or you can configure Network Flow Monitor with Amazon Organizations to view performance information for multiple accounts in an organization, by signing in with a management or delegated administrator account.

Network Flow Monitor is intended for network operators and application developers who want near real-time insights into network performance. In the Network Flow Monitor console in CloudWatch, you can see performance data for your resources' network traffic that has been aggregated from agents and grouped into different categories. For example, you can see data for flows between Availability Zones or between VPCs. Then, you can create monitors for specific flows that you want to see more details for and track more closely over time.

Using a monitor, you can quickly visualize packet loss and latency of your network connections over a time frame that you specify. For each monitor, Network Flow Monitor also generates a network health indicator (NHI). The NHI value informs you whether there were Amazon network issues for the network flows tracked by your monitor during the time period that you're evaluating. Using the NHI information, you can quickly decide whether to focus troubleshooting efforts on an Amazon network issue or network problems originating with your workloads.

To see an example of configuring and using Network Flow Monitor, see the following blog post: [Visualizing network performance of your Amazon Cloud workloads with Network Flow Monitor]().

What is Network Flow Monitor?

Network Flow Monitor is a feature of Amazon CloudWatch Network Monitoring. Network Flow Monitor uses fully-managed agents that you install in your Amazon workloads to return performance and availability metrics about network flows. Using Network Flow Monitor, you can access near real-time metrics, including retransmissions and data transferred, for your actual workloads. You can also identify whether an underlying Amazon network issue occurred for the network flows tracked by a monitor, by checking network health indicator (NHI) values.

Key features of Network Flow Monitor

- With Network Flow Monitor, you receive near real-time metrics for the latency and packet-loss experienced by TCP-based traffic within your VPC network, so that you can track and investigate network issues for your workload traffic.
- When your Amazon workloads experience network degradation, Network Flow Monitor helps you to determine if the problem is caused by your application workload or the underlying Amazon infrastructure. Then, you can quickly focus troubleshooting on the area where the issue is occurring.

How to use Network Flow Monitor

With Network Flow Monitor, you install lightweight agents on your instances, which collect and aggregate performance metrics. Network Flow Monitor agents analyze TCP traffic, and then export performance metrics to the Network Flow Monitor service backend.

Agents gather the following metrics for your workloads: TCP round-trip time (RTT), TCP retransmission timeouts, TCP retransmissions, and data (bytes) transferred. After you install agents on your instances, the agents detect the corresponding workloads that are hosted by the instances. The agents then generate network performance metrics and send the metrics to the Network Flow Monitor backend. Metrics are aggregated into categories such as subnets, Availability Zones, VPCs, and Amazon services.

Performance metrics for top contributors (by metric type) from all network flows that are in your Network Flow Monitor scope are shown on the **Workload insights** tab in the Amazon Web Services Management Console. By reviewing the tables and graphs of top contributors, you can determine where there might be impairments that you want to troubleshoot and which workloads you want to monitor on an ongoing basis, by creating a monitor.

With a monitor, you can monitor specific workloads more closely over time and see detailed information about specific network flows. In addition to viewing performance metrics for the top contributors for the network flows that you've selected, you can view topological information about the network hops that a network flow has traversed, to help you troubleshoot issues. In addition, Network Flow Monitor generates a network health indicator (NHI) for monitors. An

NHI value of **Degraded** indicates that there were Amazon network issues for at least one of the network flows tracked by your monitor, during the time period that you've selected.

In addition to reviewing the information in monitors that you create, we recommend that you also check back regularly to review metrics on the **Workload insights** page, to see the latest top contributors for performance metrics for your network flows. As you review the latest information, consider if it would be helpful to add or remove workloads from your current monitors, or create new monitors.

Contents

Supported Amazon Web Services Regions for Network Flow Monitor

Network Flow Monitor is currently available in the following Amazon Web Services Regions:

Region name	Region
Asia Pacific (Mumbai)	ap-south-1
Asia Pacific (Osaka)	ap-northeast-3
Asia Pacific (Seoul)	ap-northeast-2
Asia Pacific (Singapore)	ap-southeast-1
Asia Pacific (Sydney)	ap-southeast-2
Asia Pacific (Tokyo)	ap-northeast-1
Canada (Central)	ca-central-1
Europe (Frankfurt)	eu-central-1
Europe (Ireland)	eu-west-1

Region name	Region
Europe (London)	eu-west-2
Europe (Paris)	eu-west-3
Europe (Stockholm)	eu-north-1
South America (São Paulo)	sa-east-1
US East (N. Virginia)	us-east-1
US East (Ohio)	us-east-2
US West (N. California)	us-west-1
US West (Oregon)	us-west-2

Components and features of Network Flow Monitor

Network Flow Monitor uses or references the following concepts.

Agents

An *agent* in Network Flow Monitor is a software application that you install on your Amazon EC2 instance resources. The application has two parts:

- The first part receives events related to TCP connections and is registered within the Linux kernel using eBPF. eBPF is the Linux extended Berkley Packet Filter (eBPF) capability that allows a designated program to receive certain events raised by the Linux kernel.
- The second part aggregates the statistics collected by the eBPF portion. The agent sends the aggregated metrics to the Network Flow Monitor backend about every 30 seconds, with a 5 second potential jitter (in other words, 25 to 35 seconds).

For more information about agents, see [How it works](#).

Top contributors

Top contributors are the network flows that have the highest values for a specific metric (such as retransmissions) in your Network Flow Monitor scope or among the network flows you're tracking in a monitor. Reviewing the flows with the highest reported numbers for performance metric measurements can help you see where there might be impairments to investigate. Network Flow Monitor returns performance metrics for top contributors in your monitoring scope for *workload insights*. In addition, if you create a monitor, Network Flow Monitor returns performance metrics for top contributors for the network flows that you choose for the monitor.

Local and remote resources

A *local resource*, in a bi-directional flow of a workload, is the host where the agent is installed. For example, if a workload consists of an interaction between a web service and a backend database (for example, Amazon RDS), the EC2 instance hosting the web service, which also runs the agent, is the local resource. A local resource can be a subnet, a VPC, or an Availability Zone. The local resource is identified by the IP address and the transport protocol port, at a minimum.

A *remote resource* is the other endpoint in the bi-directional flow of a workload. In this example of a web service with a backend RDS database, Amazon RDS is the remote resource. A remote resource can be a subnet, a VPC, an Availability Zone, or an Amazon service. Just like a local resource, a remote resource is identified by the IP address of the endpoint and the transport protocol port.

Workload insights

Workload insights includes the performance metrics returned for all the network flows in your scope. In the Amazon Web Services Management Console, the **Workload insights** page provides performance data about workloads where you've installed Network Flow Monitor agents on workload instances. The **Workload insights** page provides a view into your applications that includes the amount of data transferred and several other metrics, grouped into categories of workloads. For example, you can see all the metrics for workloads with traffic between Availability Zones (AZs) or within an AZ. By using these insights, you can select workloads for which you want to create a monitor to see more details and to track network performance on an ongoing basis.

Monitors

You create a *monitor* so that you can monitor, on an ongoing basis, the network performance for one or several specific workloads, and see more detailed information about the network flows. For each monitor, Network Flow Monitor publishes end-to-end performance metrics, and a network health indicator (NHI), which you can use to help determine attribution for impairments. We recommend that you review information on the **Workload insights** page to see which network flows you want to focus on, and then create a monitor for those flows. Then, by regularly reviewing **Workload insights**, you can decide if you have the monitors that you need, or if creating new monitors would be helpful.

Network health indicator (NHI)

The *network health indicator* (NHI) is a binary value that informs you whether there were Amazon network issues for one or more of the network flows tracked by a monitor, during a time period that you choose. When the NHI value is 1, or **Degraded**, there was an Amazon network issue for at least one network flow. With the NHI indicator, you can quickly decide whether to focus troubleshooting efforts on an Amazon network issue or network problems originating with your workloads.

For more information about agents, see View Network Flow Monitor metrics in CloudWatch.

Scope

In Network Flow Monitor, the *scope* is the account or accounts that you have observability for when you look at network performance indicators. If you sign in as a management account and configure Amazon Organizations with CloudWatch, you can set your scope to more than one account in your organization (up to 100 accounts total). Otherwise, if you sign in with an Amazon Web Services account that does not have management permissions in Organizations, or if you have not configured Organizations with CloudWatch, Network Flow Monitor sets your scope to the account that you're signed in with.

Network Flow Monitor generates a unique **scope ID** for the scope. Queries for metrics data use the scope ID to determine the resources that the Network Flow Monitor generates metrics for. (You must install agents to gather and submit metrics data before you can view the performance metrics for an account with Network Flow Monitor.)

Query ID

Network Flow Monitor generates a unique *query ID* for each query that is created to retrieve performance metrics data, such as a query for top contributors for a monitor. By using a query ID with an API call in Network Flow Monitor, you can check the status of a query, stop a query, run the query again, or work with the query in other ways.

Performance metrics

Network Flow Monitor gathers and calculates end-to-end *performance metrics*, including TCP round-trip time (RTT), TCP retransmissions, TCP retransmission time outs, and bytes transferred for each flow that is in your Network Flow Monitor scope. The service aggregates these metrics and returns them to the service backend. You can view top contributors by metric type. When you see an anomaly in Network Flow Monitor, you can also check the network health indicator (NHI) to see if there is an underlying Amazon network issue.

Be aware that RTT data can be sparse because RTT is not always calculated.

You can also use Amazon CloudWatch features to create dashboards, alarms, and notifications based on these metrics. For example, you can learn about setting up alarms with Network Flow Monitor metrics by reviewing the information in [Create alarms with Network Flow Monitor].

How it works

This section provides information about how agents in Network Flow Monitor work.

How Network Flow Monitor agents work

Agents in Network Flow Monitor are installed on Amazon EC2 instances, where they gather performance metrics and send them to the Network Flow Monitor backend. Agents do not have access to the payload of your TCP connections. Agents receive only what is called the "bpf_sock_ops" structure from the Linux kernel. This structure provides the local and remote IP address and the source and destination TCP port, as well as counters and round-trip times. For list of the TCP statistics collected and published by the agent, see [View Network Flow Monitor metrics in CloudWatch].

The agent uses the Network Flow Monitor `Publish` API to send metrics to the Network Flow Monitor backend server.

Pricing for Network Flow Monitor

With Network Flow Monitor, there are no upfront costs or long-term commitments. Pricing for Network Flow Monitor has two components: resources monitored (agents installed and actively sending data) and CloudWatch metrics vended. Note that you are also charged standard CloudWatch prices for any additional metrics, dashboards, alarms, or insights that you create.

For more information about Network Flow Monitor and Amazon CloudWatch pricing, see Network Monitoring on the [Amazon CloudWatch pricing] page.

Get started with Network Flow Monitor

To help you get started, the section provides a high level overview of the steps to configure, and then gain insights, with Network Flow Monitor. For details, see the additional sections in this guide about initializing Network Flow Monitor, configuring agents, and creating monitors.

- Initialize Network Flow Monitor, to accept service-linked role permissions, create a *scope* for monitoring in Network Flow Monitor, and create an initial topology. If you want to observe network performance for network flows for instances in multiple accounts, you must integrate with Amazon Organizations, and then add the accounts to your scope. To learn more, see Initialize Network Flow Monitor.
- Deploy *agents* on your instances, by using Amazon Systems Manager or by configuring Kubernetes, depending on how your resources are deployed. If you install agents on VPC EC2 instances, make sure that you enable permissions for agents on each instance to send metrics to the Network Flow Monitor backend. To learn more, see Install Network Flow Monitor agents on instances.
- Review top contributor metrics for network flows returned by the agents, to gain *workload insights*. Workload insights provide a high-level view of the performance for network flows in the scope you're monitoring.
- Based on the network flows that you want to see detailed network information about, create one or more *monitors*. Using a monitor, you can see details metrics and information, as well as view topologies for specific network flows, over time periods that you select.
- On a regular basis:
  - Review network flow information in the monitors that you've created, to learn about and help troubleshoot network impairments in your workloads.
  - Review workload insights for the network flows that you're monitoring, to determine if the monitors that you've created are covering the most relevant network flows or if it would be helpful to create new monitors.

Network Flow Monitor API operations

The following table lists Network Flow Monitor API operations that you can use with Network Flow Monitor. The table also includes links to relevant documentation.

Action	API operation	More information
Create a flow monitor.	See CreateMonitor	See Create a monitor in Network Flow Monitor
Generate metrics for a scope of resources.	See CreateScope	See Evaluate network flows with workload insights
Remove a monitor.	See DeleteMonitor	See Delete a monitor in Network Flow Monitor
Delete a defined scope.	See DeleteScope	See Delete a monitor in Network Flow Monitor
Get information about a monitor.	See GetMonitor	See Monitor and analyze network flows using Network Flow Monitor performance metrics
Get query data for the top contributors in a specific monitor.	See GetQueryResultsMonitorTopContributors	See Monitor and analyze network flows using Network Flow Monitor performance metrics
Query the top contributors for a defined scope for workload insights.	See GetQueryResultsWorkloadInsightsTopContributors	See Evaluate network flows with workload insights
Query workload insights data for the top contributors in a specific scope.	See GetQueryResultsWorkloadInsightsTopContributorsData	See Evaluate network flows with workload insights
Check the status of a query for top contributors in a monitor to ensure that it has succeeded before reviewing the results.	See GetQueryStatusMonitorTopContributors	N/A

Action	API operation	More information
Check the status of a query on workload insights for top contributors to ensure that it has succeeded before reviewing the results.	See GetQueryStatusWorkloadInsightsTopContributors	N/A
Check the status of a query on workload insights data for top contributors to ensure that it has succeeded before reviewing the results.	See GetQueryStatusWorkloadInsightsTopContributorsData	N/A
Get information about an account, or scope, including name, status, tags, and target details.	See GetScope	See Monitor and analyze network flows using Network Flow Monitor performance metrics
List all monitors in an account.	See ListMonitors	Monitor and analyze network flows using Network Flow Monitor performance metrics
List all scopes for an account.	See ListScopes	N/A
List all tags for a specific resource.	See ListTagsForResource	See Monitor and analyze network flows using Network Flow Monitor performance metrics
See query data for top contributors in a monitor.	See StartQueryMonitorTopContributors	See Monitor and analyze network flows using Network Flow Monitor performance metrics
Start a query to gather workload insights data for top contributors.	See StartQueryWorkloadInsightsTopContributors	See Evaluate network flows with workload insights

Action	API operation	More information
Stop a query for top contributors in a monitor.	See StopQueryMonitorTopContributors	N/A
Stop a query on workload insights data for top contributors.	See StopQueryWorkloadInsightsTopContributors	N/A
Stop a query on workload insights data for top contributors.	See StopQueryWorkloadInsightsTopContributorsData	N/A
Add a tag to a resource.	See TagResource	See Edit a monitor in Network Flow Monitor
Remove a tag from a resource.	See UntagResource	See Edit a monitor in Network Flow Monitor
Update a monitor to add or remove local or remote resources.	See UpdateMonitor	See Edit a monitor in Network Flow Monitor
Modify a scope to add or remove resources that Network Flow Monitor will generate metrics on.	See UpdateScope	N/A

Examples of using the CLI with Network Flow Monitor

This section includes examples for using the Amazon Command Line Interface with Network Flow Monitor operations.

Before you begin, make sure that you log in to use the Amazon CLI with the Amazon account that provides the scope that you want to use to monitor network flows. For more information about using API actions with Network Flow Monitor, see the Network Flow Monitor API Reference Guide.

Topics

Create a monitor

To create a monitor with the Amazon CLI, use the `create-monitor` command. The following example creates a monitor named demo in the specified account.

```
aws networkflowmonitor create-monitor \
        --monitor-name demo \
        --local-resources type="AWS::EC2::VPC",identifier="arn:aws:ec2:us-
east-1:111122223333:vpc/vpc-11223344556677889"  \
        --scope-arn arn:aws:networkflowmonitor:us-east-1:111122223333:scope/sample-
aaaa-bbbb-cccc-44556677889
```

Output:

```
{
        "monitorArn": "arn:aws:networkflowmonitor:us-east-1:111122223333:monitor/demo",
        "monitorName": "demo",
        "monitorStatus": "ACTIVE",
        "tags": {}
```

```
    }
```

For more information, see Create a monitor in Network Flow Monitor.

View monitor details

To view information about a monitor with the Amazon CLI, use the `get-monitor` command.

```
aws networkflowmonitor get-monitor --monitor-name "TestMonitor"
```

Output:

```
{
    "ClientLocationType": "city",
    "CreatedAt": "2022-09-22T19:27:47Z",
    "ModifiedAt": "2022-09-22T19:28:30Z",
    "MonitorArn": "arn:aws:networkflowmonitor:us-east-1:111122223333:monitor/
TestMonitor",
    "MonitorName": "TestMonitor",
    "ProcessingStatus": "OK",
    "ProcessingStatusInfo": "The monitor is actively processing data",
    "Resources": [
        "arn:aws:ec2:us-east-1:111122223333:vpc/vpc-11223344556677889"
    ],
    "MaxCityNetworksToMonitor": 10000,
    "Status": "ACTIVE"
}
```

Create a scope

The following `create-scope` example creates a scope that is the set of resources for which Network Flow Monitor will generate network traffic metrics.

```
aws networkflowmonitor create-scope \
        --targets '[{"targetIdentifier":{"targetId":
{"accountId":"111122223333"},"targetType":"ACCOUNT"},"region":"us-east-1"}]'
```

Output:

```

    {
```

```
        "scopeId": "sample-aaaa-bbbb-cccc-11112222333",
        "status": "IN_PROGRESS",
        "tags": {}
    }
```

For more information, see Components and features of Network Flow Monitor.

Delete a monitor

The following `delete-monitor` example deletes a monitor named `Demo` in your account.

```
aws networkflowmonitor delete-monitor \
        --monitor-name Demo
```

This command produces no output.

For more information, see Delete a monitor in Network Flow Monitor.

Delete a scope

The following `delete-scope` example deletes the specified scope.

```
aws networkflowmonitor delete-scope \
        --scope-id sample-aaaa-bbbb-cccc-44556677889
```

This command produces no output.

For more information, see Components and features of Network Flow Monitor.

Get information about a monitor

The following `get-monitor` example displays information about the monitor named `demo` in the specified account.

```
aws networkflowmonitor get-monitor \
        --monitor-name Demo
```

Output:

```
{
        "monitorArn": "arn:aws:networkflowmonitor:us-east-1:111122223333:monitor/Demo",
```

```
        "monitorName": "Demo",
        "monitorStatus": "ACTIVE",
        "localResources": [
            {
                "type": "AWS::EC2::VPC",
                "identifier": "arn:aws:ec2:us-east-1:111122223333:vpc/
vpc-11223344556677889"
            }
        ],
        "remoteResources": [],
        "createdAt": "2024-12-09T12:21:51.616000-06:00",
        "modifiedAt": "2024-12-09T12:21:55.412000-06:00",
        "tags": {}
    }
```

For more information, see Components and features of Network Flow Monitor.

Retrieve data on a specific queries

The following sections provide example CLI commands to retrieve query statuses.

get-query-results-workload-insights-top-contributors-data

The `get-query-results-workload-insights-top-contributors-data` example returns the data for the specified query.

```
aws networkflowmonitor get-query-results-workload-insights-top-contributors-data \
        --scope-id sample-aaaa-bbbb-cccc-11112222333 \
        --query-id sample-dddd-eeee-ffff-44556677889
```

Output:

```
{
        "datapoints": [
            {
                "timestamps": [
                    "2024-12-09T19:00:00+00:00",
                    "2024-12-09T19:05:00+00:00",
                    "2024-12-09T19:10:00+00:00"
                ],
                "values": [
                    259943.0,
```

```
                    194856.0,
                    216432.0
                ],
                "label": "use1-az6"
            }
        ],
        "unit": "Bytes"
    }
```

get-query-results-workload-insights-top-contributors

The following `get-query-results-workload-insights-top-contributors` example returns the data for the specified query.

```
aws networkflowmonitor get-query-results-workload-insights-top-contributors \
        --scope-id sample-aaaa-bbbb-cccc-11112222333 \
        --query-id sample-dddd-eeee-ffff-44556677889
```

Output:

```
{
        "topContributors": [
            {
                "accountId": "111122223333",
                "localSubnetId": "subnet-SAMPLE1111",
                "localAz": "use1-az6",
                "localVpcId": "vpc-SAMPLE2222",
                "localRegion": "us-east-1",
                "remoteIdentifier": "",
                "value": 333333,
                "localSubnetArn": "arn:aws:ec2:us-east-1:111122223333:subnet/
subnet-2222444455556666",
                "localVpcArn": "arn:aws:ec2:us-east-1:111122223333:vpc/
vpc-11223344556677889"
            }
        ]
    }
```

get-query-status-monitor-top-contributors

The following `get-query-status-monitor-top-contributors` example displays the current status of the query in the specified account.

```
aws networkflowmonitor get-query-status-monitor-top-contributors \
        --monitor-name Demo \
        --query-id sample-dddd-eeee-ffff-44556677889
```

Output:

```
{
        "status": "SUCCEEDED"
    }
```

get-query-status-workload-insights-top-contributors-data

The following `get-query-status-workload-insights-top-contributors-data` example displays the current status of the query in the specified account.

```
aws networkflowmonitor get-query-status-workload-insights-top-contributors-data \
        --scope-id sample-aaaa-bbbb-cccc-11112222333 \
        --query-id sample-dddd-eeee-ffff-44556677889
```

Output:

```
{
        "status": "SUCCEEDED"
    }
```

get-query-results-workload-insights-top-contributors

The following `get-query-results-workload-insights-top-contributors` example displays the current status of the query in the specified account.

```
aws networkflowmonitor get-query-status-workload-insights-top-contributors \
        --scope-id sample-aaaa-bbbb-cccc-11112222333 \
        --query-id sample-dddd-eeee-ffff-44556677889
```

Output:

```
{
        "status": "SUCCEEDED"
    }
```

For more information, see Evaluate network flows with workload insights.

See scope information

The following `get-scope` example displays information about a scope, such as status, tags, name, and target details.

```
aws networkflowmonitor get-scope \
        --scope-id sample-aaaa-bbbb-cccc-11112222333
```

Output:

```
{
        "scopeId": "sample-aaaa-bbbb-cccc-11112222333",
        "status": "SUCCEEDED",
        "scopeArn": "arn:aws:networkflowmonitor:us-east-1:111122223333:scope/sample-
aaaa-bbbb-cccc-11112222333",
        "targets": [
            {
                "targetIdentifier": {
                    "targetId": {
                        "accountId": "111122223333"
                    },
                    "targetType": "ACCOUNT"
                },
                "region": "us-east-1"
            }
        ],
        "tags": {}
    }
```

For more information, see Components and features of Network Flow Monitor.

See a list of monitors for an account

The following `list-monitors` example returns all the monitors in the specified account.

```
aws networkflowmonitor list-monitors
```

Output:

```
{
```

```
        "monitors": [
            {
                "monitorArn": "arn:aws:networkflowmonitor:us-
east-1:111122223333:monitor/Demo",
                "monitorName": "Demo",
                "monitorStatus": "ACTIVE"
            }
        ]
    }
```

For more information, see Components and features of Network Flow Monitor.

See a list of scopes for an account

The following `list-scopes` example lists all the scopes in the specified account.

```
aws networkflowmonitor list-scopes
```

Output:

```
{
        "scopes": [
            {
                "scopeId": "sample-aaaa-bbbb-cccc-11112222333",
                "status": "SUCCEEDED",
                "scopeArn": "arn:aws:networkflowmonitor:us-east-1:111122223333:scope/
sample-aaaa-bbbb-cccc-11112222333"
            }
        ]
    }
```

For more information, see Components and features of Network Flow Monitor.

See the list of tags for a monitor

The following `list-tags-for-resource` example returns all the tags associated with the specified resource.

```
aws networkflowmonitor list-tags-for-resource \
        --resource-arn arn:aws:networkflowmonitor:us-east-1:111122223333:monitor/Demo
```

Output:

```
{
        "tags": {
            "Value": "Production",
            "Key": "stack"
        }
    }
```

For more information, see Tagging your Amazon CloudWatch resources.

Starting and stopping queries

The following sections provide example CLI commands for starting and stopping queries in Network Flow Monitor.

start-query-monitor-top-contributors

The following `start-query-monitor-top-contributors` example starts the query which returns a queryId to retrieve the top contributors.

```
aws networkflowmonitor start-query-monitor-top-contributors \
        --monitor-name Demo \
        --start-time 2024-12-09T19:00:00Z \
        --end-time 2024-12-09T19:15:00Z \
        --metric-name DATA_TRANSFERRED \
        --destination-category UNCLASSIFIED
```

Output:

```
{
        "queryId": "sample-dddd-eeee-ffff-44556677889"
    }
```

For more information, see Evaluate network flows with workload insights.

start-query-workload-insights-top-contributors-data

The following `start-query-workload-insights-top-contributors-data` example starts the query which returns a queryId to retrieve the top contributors.

```
aws networkflowmonitor start-query-workload-insights-top-contributors-data \
        --scope-id sample-aaaa-bbbb-cccc-11112222333 \
        --start-time 2024-12-09T19:00:00Z \
        --end-time 2024-12-09T19:15:00Z \
        --metric-name DATA_TRANSFERRED \
        --destination-category UNCLASSIFIED
```

Output:

```
{
        "queryId": "sample-dddd-eeee-ffff-44556677889"
    }
```

For more information, see Evaluate network flows with workload insights.

start-query-workload-insights-top-contributors

The following `start-query-workload-insights-top-contributors` example starts the query which returns a queryId to retrieve the top contributors.

```
aws networkflowmonitor start-query-workload-insights-top-contributors \
        --scope-id sample-aaaa-bbbb-cccc-11112222333 \
        --start-time 2024-12-09T19:00:00Z \
        --end-time 2024-12-09T19:15:00Z \
        --metric-name DATA_TRANSFERRED \
        --destination-category UNCLASSIFIED
```

Output:

```
{
        "queryId": "sample-dddd-eeee-ffff-44556677889"
    }
```

For more information, see Evaluate network flows with workload insights.

stop-query-monitor-top-contributors

The following `stop-query-monitor-top-contributors` example stops the query in the specified account.

```
aws networkflowmonitor stop-query-monitor-top-contributors \
        --monitor-name Demo \
        --query-id sample-dddd-eeee-ffff-44556677889
```

This command produces no output.

For more information, see [Evaluate network flows with workload insights].

stop-query-workload-insights-top-contributors-data

The following `stop-query-workload-insights-top-contributors-data` stops the query in the specified account.

```
aws networkflowmonitor stop-query-workload-insights-top-contributors-data \
        --scope-id sample-aaaa-bbbb-cccc-11112222333 \
        --query-id sample-dddd-eeee-ffff-44556677889
```

This command produces no output.

For more information, see [Evaluate network flows with workload insights].

stop-query-workload-insights-top-contributors

The following `stop-query-workload-insights-top-contributors` example stops the query in the specified account.

```
aws networkflowmonitor stop-query-workload-insights-top-contributors \
        --scope-id sample-aaaa-bbbb-cccc-11112222333 \
        --query-id sample-dddd-eeee-ffff-44556677889
```

This command produces no output.

For more information, see [Evaluate network flows with workload insights].

Tag a monitor

The following `tag-resource` adds a tag to the monitor in the specified account.

```
aws networkflowmonitor tag-resource \
        --resource-arn arn:aws:networkflowmonitor:us-east-1:111122223333:monitor/Demo \
```

```
        --tags Key=stack,Value=Production
```

This command produces no output.

For more information, see Tagging your Amazon CloudWatch resources.

Remove a tag from a monitor

The following `untag-resource` example removes a tag to the monitor in the specified account.

```
aws networkflowmonitor untag-resource \
        --resource-arn arn:aws:networkflowmonitor:us-east-1:111122223333:monitor/Demo \
        --tag-keys stack
```

This command produces no output.

For more information, see Tagging your Amazon CloudWatch resources.

Update an existing monitor

The following `update-monitor` example updates the monitor named ``Demo`` in the specified account.

```
aws networkflowmonitor update-monitor \
        --monitor-name Demo \
        --local-resources-to-add type="AWS::EC2::VPC",identifier="arn:aws:ec2:us-
east-1:111122223333:vpc/vpc-11223344556677889"
```

Output:

```
{
        "monitorArn": "arn:aws:networkflowmonitor:us-east-1:111122223333:monitor/Demo",
        "monitorName": "Demo",
        "monitorStatus": "ACTIVE",
        "tags": {
            "Value": "Production",
            "Key": "stack"
        }
    }
```

For more information, see Components and features of Network Flow Monitor.

Install Network Flow Monitor agents on instances

To provide performance metrics for network flows in your Amazon workloads, Network Flow Monitor relies on *agents* that you install, which send the metrics to Network Flow Monitor. You install Network Flow Monitor agents on your instances, and then set the correct permissions for the agents so that they can send metrics to the Network Flow Monitor backend.

An agent is a lightweight software application that you install on your resources, such as your VPC EC2 instances. Agents send performance metrics to the Network Flow Monitor backend on an ongoing basis. Then, you can view the metrics on the **Workload insights** page in the Network Flow Monitor console. You can also track detailed metrics for a specific network flow, or set of flows, by creating a monitor.

The instances that you install agents on must be running supported versions and distributions of Linux. Network Flow Monitor supports agents to run only on Linux, and the Linux kernel version must be 5.8 or later. The following Linux distributions are supported. Note that agents are tested to run on the latest versions of these distributions.

- Amazon Linux
- Ubuntu
- Red Hat
- Suse Linux
- Debian distributions for both x86 and aarch64

You can establish a private connection between your VPC and Network Flow Monitor agents by using Amazon PrivateLink. For more information, see Using CloudWatch, CloudWatch Synthetics, and CloudWatch Network Monitoring with interface VPC endpoints.

The steps that you follow to deploy agents in your instances depend on the type of instance: VPC EC2 instances, Amazon EKS Kubernetes instances, or self-managed (non-EKS) Kubernetes instances.

Contents

Install and manage agents for EC2 instances

Follow the steps in this section to install Network Flow Monitor agents for workloads on Amazon EC2 instances. You can install agents by using SSM or by downloading and installing prebuilt packages for the Network Flow Monitor agent by using the command line.

Regardless of the method that you use to install agents on EC2 instances, you must configure permissions for the agents to enable them to send performance metrics to the Network Flow Monitor backend.

Contents

Configure permissions for agents

To enable agents to send metrics to the Network Flow Monitor ingestion backend, the EC2 instances that the agents run in must use a role that has a policy attached with the correct permissions. To provide the required permissions, use a role that has the following Amazon managed policy attached: CloudWatchNetworkFlowMonitorAgentPublishPolicy. Attach this policy to the IAM roles of the EC2 instances where you plan to install Network Flow Monitor agents.

We recommend that you add the permissions before you install agents on the EC2 instances. You can choose to wait until after you install agents, but the agents won't be able to send metrics to the service until the permissions are in place.

To add permissions for Network Flow Monitor agents

1. In the Amazon Web Services Management Console, in the Amazon EC2 console, locate the EC2 instances that you plan to install Network Flow Monitor agents on.
2. Attach the CloudWatchNetworkFlowMonitorAgentPublishPolicy to the IAM role for each instance.

   If an instance doesn't have an IAM role attached, choose a role by doing the following:

   1. Under **Actions**, choose **Security**.
   2. Choose **Modify IAM role**, or create a new role by choosing **Create new IAM role**.

3. Choose a role for the instance, and attach the CloudWatchNetworkFlowMonitorAgentPublishPolicy policy.

Install agents on EC2 instances with SSM

Network Flow Monitor agents provide performance metrics about network flows. Follow the steps in this section to install and work with Network Flow Monitor agents on EC2 instances, by using Amazon Systems Manager. If you use Kubernetes, skip to the next sections for information about installing agents with Amazon EKS clusters or self-managed Kubernetes clusters.

Network Flow Monitor provides a Distributor package for you in Systems Manager to use to install or uninstall agents. In addition, Network Flow Monitor provides a document to activate or deactivate agents, by using the Document Type command. Use standard Systems Manager procedures to use the package and the document, or follow the steps provided here for detailed guidance.

For more information in general about using Systems Manager, see the following documentation:

- Amazon Systems Manager Run Command
- Amazon Systems Manager Distributor

Complete the steps in the following sections to configure permissions, install, and work with Network Flow Monitor agents.

Contents

Install or uninstall agents by using Systems Manager

Network Flow Monitor provides a distributor package in Amazon Systems Manager for you to install Network Flow Monitor agents: **AmazonCloudWatchNetworkFlowMonitorAgent**. To access and run the package to install agents, follow the steps provided here.

To install agents in EC2 instances

1. In the Amazon Web Services Management Console, in Amazon Systems Manager, under **Node Tools**, choose **Distributor**.

2. Under **Owned by Amazon**, locate the Network Flow Monitor package, **AmazonCloudWatchNetworkFlowMonitorAgent**, and select it.
3. In the **Run command** flow, choose **Install one time** or **Install on schedule**.
4. In the **Target selection** section, choose how you want to select your EC2 instances to install agents on. You can select instances based on tags, choose instances manually, or base the choice on resource groups.
5. In the **Commmand parameters** section, under **Action**, choose **Install**.
6. Scroll down, if necessary, and then choose **Run** to start the installation.

If the installation is successful and the instances have permissions to access Network Flow Monitor endpoints, the agent will start collecting metrics and send reports to the Network Flow Monitor backend.

Agents that are active (sending metrics data) incur billing costs. For more information about Network Flow Monitor and Amazon CloudWatch pricing, see Network Monitoring on the Amazon CloudWatch pricing page. If you don't need metrics data temporarily, you can deactivate an agent. For more information, see Activate or deactivate agents. If you no longer need Network Flow Monitor agents, you can uninstall them from the EC2 instances.

To uninstall agents from EC2 instances

1. In the Amazon Web Services Management Console, in Amazon Systems Manager, under **Node Tools**, choose **Distributor**.
2. Under **Owned by Amazon**, locate the Network Flow Monitor package, **AmazonCloudWatchNetworkFlowMonitorAgent**, and select it.
3. In the **Commmand parameters** section, under **Action**, choose **Uninstall**.
4. Select the EC2 instances to uninstall agents from.
5. Scroll down, if necessary, and then choose **Run** to start the installation.

Activate or deactivate agents by using Systems Manager

After you install a Network Flow Monitor agent with SSM, you must activate it to receive network flow metrics from the instance where it's installed. Agents that are active (sending metrics data) incur billing costs. For more information about Network Flow Monitor and Amazon CloudWatch pricing, see Network Monitoring on the Amazon CloudWatch pricing page. If you don't need metrics data temporarily, you can deactivate an agent to prevent ongoing billing for the agent.

Network Flow Monitor provides a document in Amazon Systems Manager that you can use activate or deactivate agents that you've installed on your EC2 instances. By running this document to manage the agents, you can activate them to begin receiving performance metrics. Or, you can deactivate them to temporarily stop metrics from being sent,without uninstalling the agents.

The document in SSM that you can use to activate or deactivate agents is called **AmazonCloudWatch-NetworkFlowMonitorManageAgent**. To access and run the document, follow the steps in the procedure.

To activate or deactivate Network Flow Monitor agents

1. In the Amazon Web Services Management Console, in Amazon Systems Manager, under **Change Management Tools**, choose **Documents**.
2. Under **Owned by Amazon**, locate the Network Flow Monitor document, **AmazonCloudWatch-NetworkFlowMonitorManageAgent**, and select the document.
3. In the **Target selection** section, choose how you want to select your EC2 instances to install agents on. You can select instances based on tags, choose instances manually, or base the choice on resource groups.
4. In the **Command parameters** section, under **Action**, choose **Activate** or **Deactivate**, depending on the action that you want to take for the agents.
5. Scroll down, if necessary, and then choose **Run** to start the installation.

Download prebuilt packages of the Network Flow Monitor agent by using the command line

You can use the command line to install the Network Flow Monitor agent as a package in Amazon Linux 2023, or download and install prebuilt packages of the Network Flow Monitor agent.

Before or after you download a prebuilt package, you can optionally verify the package signature. For more information, see [Verify the signature of the Network Flow Monitor agent package].

Choose from the following instructions, depending on the Linux operating system that you're using and the type of installation that you want.

Amazon Linux AMIs

The Network Flow Monitor agent is available as a package in Amazon Linux 2023. If you're using this operating system, you can install the package by entering the following command:

```
sudo yum install network-flow-monitor-agent
```

You must also make sure that the IAM role attached to the instance has the CloudWatchNetworkFlowMonitorAgentPublishPolicy policy attached. For more information, see Configure permissions for agents.

Amazon Linux 2023

Install the package for your architecture by using one of the following commands:

- **x86_64**: `sudo yum install https://networkflowmonitoragent.awsstatic.com/latest/x86_64/network-flow-monitor-agent.rpm`
- **ARM64 (Graviton)**: `sudo yum install https://networkflowmonitoragent.awsstatic.com/latest/arm64/network-flow-monitor-agent.rpm`

Verify that Network Flow Monitor agent is successfully installed by running the following command and verifying that the response shows that the agent is enabled and active:

```
service network-flow-monitor status
network-flow-monitor.service - Network Flow Monitor Agent
     Loaded: loaded (/usr/lib/systemd/system/network-flow-monitor.service; enabled;
 preset: enabled)
     Active: active (running) since Wed 2025-04-23 19:17:16 UTC; 1min 9s ago
```

DEB-based distributions (Debian, Ubuntu)

Install the package for your architecture by using one of the following commands:

- **x86_64**: `wget https://networkflowmonitoragent.awsstatic.com/latest/x86_64/network-flow-monitor-agent.deb`
- **ARM64 (Graviton)**: `wget https://networkflowmonitoragent.awsstatic.com/latest/arm64/network-flow-monitor-agent.deb`

Install the package by using the following command: `$ sudo apt-get install ./network-flow-monitor-agent.deb`

Verify that Network Flow Monitor agent is successfully installed by running the following command and verifying that the response shows that the agent is enabled and active:

```
service network-flow-monitor status
network-flow-monitor.service - Network Flow Monitor Agent
```

```
    Loaded: loaded (/usr/lib/systemd/system/network-flow-monitor.service; enabled;
preset: enabled)
    Active: active (running) since Wed 2025-04-23 19:17:16 UTC; 1min 9s ago
```

Verify the signature of the Network Flow Monitor agent package

The Network Flow Monitor agent rpm and deb installer packages for Linux instances are cryptographically signed. You can use a public key to verify that the agent package is original and unmodified. If the files are damaged or have been altered, the verification fails. You can verify the signature of the installer package using either RPM or GPG. The following information is for Network Flow Monitor agent versions 0.1.3 or later.

To find the correct signature file for each architecture and operating system, use the following table.

Architecture	Platform	Download link	Signature file link
x86-64	Amazon Linux 2023	https://networkflowmonitoragent.awsstatic.com/latest/x86_64/network-flow-monitor-agent.rpm	https://networkflowmonitoragent.awsstatic.com/latest/x86_64/network-flow-monitor-agent.rpm.sig
ARM64	Amazon Linux 2023	https://networkflowmonitoragent.awsstatic.com/latest/arm64/network-flow-monitor-agent.rpm	https://networkflowmonitoragent.awsstatic.com/latest/arm64/network-flow-monitor-agent.rpm.sig
x86-64	Debian/Ubuntu	https://networkflowmonitoragent.awsstatic.com/latest/x86_64/network-flow-monitor-agent.deb	https://networkflowmonitoragent.awsstatic.com/latest/x86_64/network-flow-monitor-agent.deb.sig
ARM64	Debian/Ubuntu	https://networkflowmonitoragent.awsstatic.com/latest/arm64/network-flow-monitor-agent.deb	https://networkflowmonitoragent.awsstatic.com/latest/arm64/network-flow-monitor-agent.deb.sig

Follow the steps here to verify the signature of the Network Flow Monitor agent.

To verify the signature of the Network Flow Monitor agent for Amazon S3 package

1. Install GnuPG so that you can run the gpg command. GnuPG is required to verify the authenticity and integrity of a downloaded Network Flow Monitor agent for an Amazon S3 package. GnuPG is installed by default on Amazon Linux Amazon Machine Images (AMIs).

2. Copy the following public key and save it to a file named `nfm-agent.gpg`.

```
-----BEGIN PGP PUBLIC KEY BLOCK-----

mQINBGf0b5IBEAC6YQc0aYrTbcHNWWMbLuqsqfspzWrtCvoU0yQ62ld7nvCGBha9
lu4lbhtiwoDawC3h6Xsxc3Pmm6kbMQfZdbo4Gda4ahf6zDOVI5zVHs3Yu2VXC2AU
5BpKQJmYddTb7dMI3GBgEodJY05NHQhq1Qd2ptdh03rsX+96Fvi4A6t+jsGzMLJU
I+hGEKGif69pJVyptJSibK5bWCDXh3eS/+vB/CbXumAKi0sq4rXv/VPiIhn6bsCI
A2lmzFd3vMJQUM/T7m7skrqetZ4mWHr1LPDFPK/H/81s8TJawx7MACsK6kIRUxu+
oicW8Icmg9S+BpIgONT2+Io5P1tYO5a9AyVF7X7gU0VgHUA1RoLnjHQHXbCmnFtW
cYEuwhUuENMl+tLQCZ+fk0kKjOlIKqeS9AVwhks92oETh8wpTwTE+DTBvUBP9aHo
S39RTiJCnUmA6ZCehepgpwW9AYCc1lHv/xcahD418E0UHV22qIw943EwAkzMDA4Q
damdRm0Nud0OmilCjo9oogEB+NUoy//5XgQMH1hhfsHquVLU/tneYexXYMfo/Iu5
TKyWL2KdkjKKP/dMR4lMAXYi0RjTJJ5tg5w/VrHhrHePFfKdYsgN6pihWwj2Px/M
ids3W1Ce50LOEBc2MOKXYXGd9OZWyR8l15ZGkySvLqVlRGwDwKGMC/nS2wARAQAB
tEJOZXR3b3JrIEZsb3cgTW9uaXRvciBBZ2VudCA8bmV0d29yay1mbG93LW1vbml0
b3ItYWdlbnRAYW1hem9uLmNvbT6JAlcEEwEIAEEWIQR2c2ypl63T6dJ3JqjvvaTM
vJX60QUCZ/RvkgIbAwUJBaOagAULCQgHAgIiAgYVCgkICwIEFgIDAQIeBwIXgAAK
CRDvvaTMvJX60euSD/9cIu2BDL4+MFFHhyHmG3/se8+3ibW0g8SyP3hsnq7qN+bm
ZzLAhll7DVoveNmEHI1VC7Qjwb30exgLcyK2Ld6uN6lwjjK0qiGGz943t230pJ3z
u7V2fVtAN+vgDVmD7agE6iqrRCWu3WfcgzFlEkE/7nkhtbWzlaK+NkdEBzNZ+W7/
FmLClzIbMjIBW2M8LdeZdQX0SWljy18x7NGNukWeNTJxmkDsjAeKl+zkXYk9h7ay
n3AVl1KrLZ5P9vQ5XsV5e4T6qfQ3XNY1lm54cpa+eD7NyYcTGRDK+vIxO4xD8i2M
yl1iNf2+84Tt6/SAgR/P9SJ5tbKD0iU9n4g1eBJVGmHDuXTtDR4H/Ur7xRSxtuMl
yZP/sLWm8p7+Ic7aQJ5OVw36MC7Oa7/K/zQEnLFFPmgBwGGiNiw5cUSyCBHNvmtv
FK0Q2XMXtBEBU9f44FMyzNJqVdPywg8Y6xE4wc/68uy7G6PyqoxDSP2ye/p+i7oi
OoA+OgifchZfDVhe5Ie0zKR0/nMEKTBV0ecjglb/WhVezEJgUFsQcjfOXNUBesJW
a9kDGcs3jIAchzxhzp/ViUBmTg6SoGKh3t+3uG/RK2ougRObJMW3G+DI7xWyY+3f
7YsLm0eDd3dAZG3PdltMGp0hKTdslvpws9qoY8kyR0Fau4l222JvYP27BK44qg==
=INr5
-----END PGP PUBLIC KEY BLOCK-----
```

3. Import the public key into your keyring and note the returned value.

```
PS>  gpg --import nfm-agent.gpg
gpg: key 3B789C72: public key "Network Flow Monitor Agent" imported
gpg: Total number processed: 1
```

```
gpg: imported: 1 (RSA: 1)
```

Make a note of the key value because you need it in the next step. In this example, the key value is 3B789C72.

4. Verify the fingerprint by running the following command. Be sure to replace *key-value* with the value from the preceding step. We recommend that you use GPG to verify the fingerprint even if you use RPM to verify the installer package.

```
PS>  gpg --fingerprint key-value
pub   rsa4096 2025-04-08 [SC] [expires: 2028-04-07]
      7673 6CA9 97AD D3E9 D277  26A8 EFBD A4CC BC95 FAD1
uid   Network Flow Monitor Agent <network-flow-monitor-agent@amazon.com>
```

The fingerprint string should be equal to the following:

`7673 6CA9 97AD D3E9 D277 26A8 EFBD A4CC BC95 FAD1`

If the fingerprint string doesn't match, don't install the agent. Contact Amazon Web Services.

After you have verified the fingerprint, you can use it to verify the signature of the Network Flow Monitor agent package.

5. Download the package signature file, if you haven't already done so, based on your instance's architecture and operating system.

6. Verify the installer package signature. Be sure to replace the `signature-filename` and `agent-download-filename` with the values that you specified when you downloaded the signature file and agent, as shown in the table earlier in this topic.

```
PS> gpg --verify sig-filename agent-download-filename
gpg: Signature made Tue Apr  8 00:40:02 2025 UTC
gpg:                using RSA key 77777777EXAMPLEKEY
gpg:                issuer "network-flow-monitor-agent@amazon.com"
gpg: Good signature from "Network Flow Monitor Agent <network-flow-monitor-
agent@amazon.com>" [unknown]
gpg: WARNING: Using untrusted key!
```

If the output includes the phrase `BAD signature`, check to make sure that you performed the procedure correctly. If you continue to get this response, contact [Amazon Support](#) and avoid using the downloaded file.

Note the warning about trust. A key is trusted only if you or someone who you trust has signed it. This doesn't mean that the signature is invalid, only that you have not verified the public key.

Next, follow the steps here to verify the RPM package.

To verify the signature of the RPM package

1. Copy the following public key and save it to a file named `nfm-agent.gpg`.

```
-----BEGIN PGP PUBLIC KEY BLOCK-----

mQINBGf0b5IBEAC6YQc0aYrTbcHNWWMbLuqsqfspzWrtCvoU0yQ62ld7nvCGBha9
lu4lbhtiwoDawC3h6Xsxc3Pmm6kbMQfZdbo4Gda4ahf6zDOVI5zVHs3Yu2VXC2AU
5BpKQJmYddTb7dMI3GBgEodJY05NHQhq1Qd2ptdh03rsX+96Fvi4A6t+jsGzMLJU
I+hGEKGif69pJVyptJSibK5bWCDXh3eS/+vB/CbXumAKi0sq4rXv/VPiIhn6bsCI
A2lmzFd3vMJQUM/T7m7skrqetZ4mWHr1LPDFPK/H/81s8TJawx7MACsK6kIRUxu+
oicW8Icmg9S+BpIgONT2+Io5P1tYO5a9AyVF7X7gU0VgHUA1RoLnjHQHXbCmnFtW
cYEuwhUuENMl+tLQCZ+fk0kKjOlIKqeS9AVwhks92oETh8wpTwTE+DTBvUBP9aHo
S39RTiJCnUmA6ZCehepgpwW9AYCc1lHv/xcahD418E0UHV22qIw943EwAkzMDA4Q
damdRm0Nud0OmilCjo9oogEB+NUoy//5XgQMH1hhfsHquVLU/tneYexXYMfo/Iu5
TKyWL2KdkjKKP/dMR4lMAXYi0RjTJJ5tg5w/VrHhrHePFfKdYsgN6pihWwj2Px/M
ids3W1Ce50LOEBc2MOKXYXGd9OZWyR8l15ZGkySvLqVlRGwDwKGMC/nS2wARAQAB
tEJOZXR3b3JrIEZsb3cgTW9uaXRvciBBZ2VudCA8bmV0d29yay1mbG93LW1vbml0
b3ItYWdlbnRAYW1hem9uLmNvbT6JAlcEEwEIAEEWIQR2c2ypl63T6dJ3JqjvvaTM
vJX60QUCZ/RvkgIbAwUJBaOagAULCQgHAgIiAgYVCgkICwIEFgIDAQIeBwIXgAAK
CRDvvaTMvJX60euSD/9cIu2BDL4+MFFHhyHmG3/se8+3ibW0g8SyP3hsnq7qN+bm
ZzLAhll7DVoveNmEHI1VC7Qjwb30exgLcyK2Ld6uN6lwjjK0qiGGz943t230pJ3z
u7V2fVtAN+vgDVmD7agE6iqrRCWu3WfcgzFlEkE/7nkhtbWzlaK+NkdEBzNZ+W7/
FmLClzIbMjIBW2M8LdeZdQX0SWljy18x7NGNukWeNTJxmkDsjAeKl+zkXYk9h7ay
n3AVl1KrLZ5P9vQ5XsV5e4T6qfQ3XNY1lm54cpa+eD7NyYcTGRDK+vIxO4xD8i2M
yl1iNf2+84Tt6/SAgR/P9SJ5tbKD0iU9n4g1eBJVGmHDuXTtDR4H/Ur7xRSxtuMl
yZP/sLWm8p7+Ic7aQJ5OVw36MC7Oa7/K/zQEnLFFPmgBwGGiNiw5cUSyCBHNvmtv
FK0Q2XMXtBEBU9f44FMyzNJqVdPywg8Y6xE4wc/68uy7G6PyqoxDSP2ye/p+i7oi
OoA+OgifchZfDVhe5Ie0zKR0/nMEKTBV0ecjglb/WhVezEJgUFsQcjfOXNUBesJW
a9kDGcs3jIAchzxhzp/ViUBmTg6SoGKh3t+3uG/RK2ougRObJMW3G+DI7xWyY+3f
7YsLm0eDd3dAZG3PdltMGp0hKTdslvpws9qoY8kyR0Fau4l222JvYP27BK44qg==
=INr5
-----END PGP PUBLIC KEY BLOCK-----
```

2. Import the public key into your keyring.

```
PS>  gpg --import nfm-agent.gpg
```

3. Verify the installer package signature. Be sure to replace the `agent-download-filename` with the value that you specified when you downloaded the agent, as shown in the table earlier in this topic.

```
PS>  rpm --checksig agent-download-filename
```

For example, for the x86_64 architecture on Amazon Linux 2023, use the following command:

```
PS>  rpm --checksig network-flow-monitor-agent.rpm
```

This command returns output similar to the following.

```
network-flow-monitor-agent.rpm: digests signatures OK
```

If the output contains the phrase `NOT OK (MISSING KEYS: (MD5) key-id)`, check to make sure that you performed the procedure correctly. If you continue to get this response, contact Amazon Support and don't install the agent.

Install agents for self-managed Kubernetes instances

Follow the steps in this section to install Network Flow Monitor agents for workloads on self-managed Kubernetes clusters. After you complete the steps, Network Flow Monitor agent pods will be running on all of your self-managed Kubernetes cluster nodes.

If you use Amazon Elastic Kubernetes Service (Amazon EKS), the installation steps to follow are in the following section: Install the EKS Amazon Network Flow Monitor Agent add-on.

Contents

Before you begin

Before you start the installation process, follow the steps in this section to make sure that your environment is set up to successfully install agents on the right Kubernetes clusters.

Ensure that your version of Kubernetes is supported

Network Flow Monitor agent installation requires Kubernetes Version 1.25, or a more recent version.

Ensure that you have installed required tools

The scripts that you use for this installation process require that you install the following tools. If you don't have the tools installed already, see the provided links for more information.

- The Amazon Command Line Interface (CLI). For more information, see Installing or updating to the latest version of the Amazon Command Line Interface in the Amazon Command Line Interface Reference Guide.
- The Helm package manager. For more information, see Installing Helm on the Helm website.
- The `kubectl` command line tool. For more information, see Install kubectl on the Kubernetes website.
- The `make` Linux command dependency. For more information, see the following blog post: Intro to make Linux Command: Installation and Usage. For example, do one of the following:
  - For Debian based distributions, such as Ubuntu, use the following command: `sudo apt-get install make`
  - For RPM-based distributions, such as CentOS, use the following command: `sudo yum install make`

Ensure that you have valid, correctly configured KubeConfig environment variables

Network Flow Monitor agent installation uses the Helm package manager tool, which uses the kubeconfig variable, `$HELM_KUBECONTEXT`, to determine the target Kubernetes clusters to work with. Also, be aware that when Helm runs installation scripts, by default, it references the standard `~/.kube/config` file. You can change the configuration environment variables, to use a different config file (by updating `$KUBECONFIG`) or to define the target cluster you want to work with (by updating `$HELM_KUBECONTEXT`).

Create a Network Flow Monitor Kubernetes namespace

The Network Flow Monitor agent's Kubernetes application installs its resources into a specific namespace. The namespace must exist for the installation to succeed. To ensure that the required namespace is in place, you can do one of the following:

- Create the default namespace, `amazon-network-flow-monitor`, before you begin.
- Create a different namespace, and then define it in the `$NAMESPACE` environment variable when you run the installation to make targets.

Download Helm charts and install agents

You can download the Network Flow Monitor agent Helm charts from the Amazon public repository by using the following command. Make sure that you first authenticate with your GitHub account.

`git clone https://github.com/aws/network-flow-monitor-agent.git`

In the `./charts/amazon-network-flow-monitor-agent` directory, you can find the Network Flow Monitor agent Helm charts and Makefile that contain the installation make targets that you use for installing agents. You install agents for Network Flow Monitor by using the following Makefile target: `helm/install/customer`

You can customize the installation if you like, for example, by doing the following:

```
# Overwrite the kubeconfig files to use
KUBECONFIG=<MY_KUBECONFIG_ABS_PATH> make helm/install/customer

# Overwrite the Kubernetes namespace to use
NAMESPACE=<MY_K8S_NAMESPACE> make helm/install/customer
```

To verify that the Kubernetes application pods for the Network Flow Monitor agents have been created and deployed successfully, check to be sure that their state is `Running`. You can check state of the agents by running the following command: `kubectl get pods -o wide -A | grep amazon-network-flow-monitor`

Configure permissions for agents to deliver metrics

After you install agents for Network Flow Monitor, you must enable the agents to send network metrics to the Network Flow Monitor ingestion APIs. Agents in Network Flow Monitor must have permission to access the Network Flow Monitor ingestion APIs so that they can deliver network flow metrics that they've collected for each instance. You grant this access by implementing IAM roles for service accounts (IRSA).

To enable agents to deliver network metrics to Network Flow Monitor, follow the steps in this section.

1. **Implement IAM roles for service accounts**

   IAM roles for service accounts provides the ability to manage credentials for your applications, similar to the way that Amazon EC2 instance profiles provide credentials to Amazon EC2 instances. Implementing IRSA is the recommended way to provide all permissions required by Network Flow Monitor agents to successfully access Network Flow Monitor ingestion APIs. For more information, see [IAM roles for service accounts] in the Amazon EKS User Guide.

   When you set up IRSA for Network Flow Monitor agents, use the following information:

   - **ServiceAccount:** When you define your IAM role trust policy, for `ServiceAccount`, specify `aws-network-flow-monitor-agent-service-account`.
   - **Namespace:** For the `namespace`, specify `amazon-network-flow-monitor`.
   - **Temporary credentials deployment:** When you configure permissions after you have deployed Network Flow Monitor agent pods, updating the `ServiceAccount` with your IAM role, Kubernetes does not deploy the IAM role credentials. To ensure that the Network Flow Monitor agents acquire the IAM role credentials that you've specified, you must rolling out a restart of `DaemonSet`. For example, use a command like the following:

     ```
     kubectl rollout restart daemonset -n amazon-network-flow-monitor aws-network-flow-monitor-agent
     ```

2. **Confirm that the Network Flow Monitor agent is successfully accessing the Network Flow Monitor ingestion APIs**

   You can check to make sure that your configuration for agents is working correctly by using the HTTP 200 logs for Network Flow Monitor agent pods. First, search for a Network Flow Monitor agent pod, and then search through the log files to find successful HTTP 200 requests. For example, you can do the following:

   a. Locate a Network Flow Monitor agent Pod name. For example, you can use the following command:

   ```
   RANDOM_AGENT_POD_NAME=$(kubectl get pods -o wide -A | grep amazon-network-flow-monitor | grep Running | head -n 1 | tr -s ' ' | cut -d " " -f 2)
   ```

b. Grep all the HTTP logs for the pod name that you've located. If you've changed the NAMESPACE, make sure that you use the new one.

```
NAMESPACE=amazon-network-flow-monitor
kubectl logs $RANDOM_AGENT_POD_NAME -\-namespace ${NAMESPACE} | grep HTTP
```

If access has been granted successfully, you should see log entries similar to the following:

```
...
{"level":"INFO","message":"HTTP request
 complete","status":200,"target":"amzn_nefmon::reports::publisher_endpoint","timestamp":17370
{"level":"INFO","message":"HTTP request
 complete","status":200,"target":"amzn_nefmon::reports::publisher_endpoint","timestamp":17370
```

Note that the Network Flow Monitor agent publishes network flow reports every 30 seconds, by calling the Network Flow Monitor ingestion APIs.

Install the EKS Amazon Network Flow Monitor Agent add-on

Follow the steps in this section to install the Amazon Network Flow Monitor Agent add-on for Amazon Elastic Kubernetes Service (Amazon EKS), to send Network Flow Monitor metrics to the Network Flow Monitor backend from Kubernetes clusters. After you complete the steps, Amazon Network Flow Monitor Agent pods will be running on all of your Kubernetes cluster nodes.

If you use self-managed Kubernetes clusters, the installation steps to follow are in the previous section: Install agents for self-managed Kubernetes instances.

Be aware that Customer Managed prefix lists Customer Managed prefix lists are not supported for Network Flow Monitor.

You can install the add-on by using the console or by using API commands with the Amazon Command Line Interface.

Contents

- Install the Amazon Network Flow Monitor Agent add-on by using the Amazon Command Line Interface

Prerequisites for installing the add-on

Regardless of whether you use the console or the CLI to install the Amazon Network Flow Monitor Agent add-on, there are several requirements for installing the add-on with Kubernetes.

Ensure that your version of Kubernetes is supported

Network Flow Monitor agent installation requires Kubernetes Version 1.25, or a more recent version.

Amazon EKS Pod Identity Agent add-on installation

You can install the Amazon EKS Pod Identity Agent add-on in the console or by using the CLI.

Amazon EKS Pod Identity associations provide the ability to manage credentials for your applications, similar to the way that Amazon EC2 instance profiles provide credentials to Amazon EC2 instances. Amazon EKS Pod Identity provides credentials to your workloads with an additional Amazon EKS Auth API and an agent pod that runs on each node.

To learn more and see the steps for installing the Amazon EKS Pod Identity add-on, see Set up the Amazon EKS Pod Identity Agent in the Amazon EKS User Guide.

Install the Amazon Network Flow Monitor Agent add-on by using the console

Follow the steps in this section to install and configure the Amazon Network Flow Monitor Agent add-on in the Amazon EKS console.

If you have already installed the add-on and have issues upgrading to a new version, see Troubleshoot issues in EKS agents installation.

Before you begin, make sure that you have installed the Amazon EKS Pod Identity Agent add-on. For more information, see the previous section.

To install the add-on using the console

1. In the Amazon Web Services Management Console, navigate to the Amazon EKS console.
2. On the page for installing add-ons, in the list of add-ons, choose **Amazon Network Flow Monitor Agent**.

3. Configure the add-on settings.

   1. For **Add-on access**, choose **EKS Pod Identity**.
   2. For the IAM role to use with the add-on, use a role that has the following Amazon managed policy attached: CloudWatchNetworkFlowMonitorAgentPublishPolicy. This policy gives permission for an agent to send telemetry reports to the Network Flow Monitor endpoint. If you don't already have a role with the policy attached, create a role by choosing **Create recommended role**, and following the steps in the IAM console.
   3. Choose **Next**.

4. On the **Review and add** page, make sure that the add-on configuration looks correct, and then choose **Create**.

Install the Amazon Network Flow Monitor Agent add-on by using the Amazon Command Line Interface

Follow the steps in this section to install the Amazon Network Flow Monitor Agent add-on for Amazon EKS by using the Amazon Command Line Interface.

1. Install the EKS Pod Identity Agent add-on

Before you begin, make sure that you have installed the Amazon EKS Pod Identity Agent add-on. For more information, see the earlier section.

2. Create the required IAM role

The Amazon Network Flow Monitor Agent add-on must have permission to send metrics to the Network Flow Monitor backend. You must attach a role with the required permissions when you create the add-on. Create a role that has the following Amazon managed policy attached: CloudWatchNetworkFlowMonitorAgentPublishPolicy. You need the ARN of this IAM role to install the add-on.

3. Install the Amazon Network Flow Monitor Agent add-on

To install the Amazon Network Flow Monitor Agent add-on for your cluster, run the following command:

```
aws eks create-addon --cluster-name CLUSTER NAME --addon-name aws-
network-flow-monitoring-agent --region Amazon REGION --pod-identity-
associations serviceAccount=aws-network-flow-monitor-agent-service-
account,roleArn=IAM ROLE ARN
```

The result should be similar to the following:

```
{
    "addon": {
        "addonName": "aws-network-flow-monitoring-agent",
        "clusterName": "ExampleClusterName",
        "status": "CREATING",
        "addonVersion": "v1.0.0-eksbuild.1",
        "health": {
            "issues": []
        },
        "addonArn": "arn:aws:eks:us-west-2:000000000000:addon/ExampleClusterName/
aws-network-flow-monitoring-agent/eec11111-bbbb-EXAMPLE",
        "createdAt": "2024-10-25T16:38:07.213000+00:00",
        "modifiedAt": "2024-10-25T16:38:07.240000+00:00",
        "tags": {},
         "podIdentityAssociations": [
            "arn:aws:eks:us-west-2:000000000000:podidentityassociation/
ExampleClusterName/a-3EXAMPLE5555555"
         ]
    }
  }
```

4. Make sure that the add-on is active

Review the installed Amazon Network Flow Monitor Agent add-on to ensure that it's active for your cluster. Run the following command to verify that the status is `ACTIVE`:

```
aws eks describe-addon --cluster-name CLUSTER NAME --addon-name aws-network-flow-monitoring-agent --region AWS REGION
```

The result should be similar to the following:

```
{
    "addon": {
        "addonName": "aws-network-flow-monitoring-agent",
        "clusterName": "ExampleClusterName",
        "status": "ACTIVE",
        "addonVersion": "v1.0.0-eksbuild.1",
        "health": {
            "issues": []
        },
```

```
        "addonArn": "arn:aws:eks:us-west-2:000000000000:addon/ExampleClusterName/
aws-network-flow-monitoring-agent/eec11111-bbbb-EXAMPLE",
        "createdAt": "2024-10-25T16:38:07.213000+00:00",
        "modifiedAt": "2024-10-25T16:38:07.240000+00:00",
        "tags": {},
         "podIdentityAssociations": [
            "arn:aws:eks:us-west-2:000000000000:podidentityassociation/
ExampleClusterName/a-3EXAMPLE5555555"
         ]
    }
  }
```

Initialize Network Flow Monitor

Before you can view performance metrics for network flows, you must initialize Network Flow Monitor, which grants required permissions and creates an initial topology for your account or accounts. If you plan to monitor resources for multiple accounts, you must also configure Amazon Organizations with Amazon CloudWatch. Then, you specify accounts for your Network Flow Monitor scope, so that Network Flow Monitor can create an initial topology for all the accounts that you'll be tracking performance metrics for.

In addition, you must install agents on your instances, to send performance metrics to the Network Flow Monitor backup ingestion server. For more information, see Install Network Flow Monitor agents on instances.

The steps that you take to initialize Network Flow Monitor vary depending on whether you are measuring performance metrics for resources in a single account, or you want to monitor metrics from resources that are owned by multiple accounts in your organization.

- Single account monitoring initialization
- Multi-account monitoring initialization

Initialize Network Flow Monitor for single account monitoring

To initialize Network Flow Monitor to monitor network performance metrics, you must grant permissions and Network Flow Monitor must create the initial topology for your account. When you monitor resources in just one account, Network Flow Monitor sets your account as the scope for network monitoring and creates a topology for that scope.

Initializing Network Flow Monitor does the following:

- Grants permissions for Network Flow Monitor to use required service-linked roles with your account. Network Flow Monitor requires you to grant it specific permissions so that the feature can send metrics to Amazon CloudWatch on your behalf, as well as create topologies of network flows. For more information, see Service-linked roles for Network Flow Monitor.
- Sets your monitoring scope for Network Flow Monitor to the Amazon account that you're signed in with. For more information, see **Scope** in Components and features of Network Flow Monitor.
- Creates an initial topology for your scope.

To initialize Network Flow Monitor by setting up the service-linked roles that provide the required permissions, setting the scope to your account, and creating a topology for network flow performance monitoring, follow these steps.

To initialize Network Flow Monitor

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the left navigation pane, under **Network Monitoring**, choose **Flow monitors**.
3. In the **Getting started with Network Flow Monitor** section, under Step 1, choose **Start initialization**.
4. On the **Configure Network Flow Monitor** page, scroll down, and then choose **Initialize Network Flow Monitor**.

Completing the initialization can take 20-30 minutes.

After you initialize Network Flow Monitor for your account, before you can view network flow performance metrics, you must also install Network Flow Monitor agents for your resources that send performance metrics to the Network Flow Monitor backed ingestion server. For more information, see Install Network Flow Monitor agents on instances.

Initialize Network Flow Monitor for multi-account monitoring

If you want to monitor network flows in Network Flow Monitor for resources that are owned by different accounts, you must first configure Amazon CloudWatch with Amazon Organizations. To use multiple accounts in Network Flow Monitor, you're required to turn on trusted access for CloudWatch, and it's a best practice to also register a delegated administrator.

In addition, if you plan to create monitors for network flows from the console, you must add a Network Flow Monitor policy to the role that is attached to your resources. The policy enables you to view resources from other accounts in the console, so that you can add the resources in multiple accounts to a monitor.

To monitor network flows for resources that are owned by different accounts, there are additional configuration steps to take. First, as the management account, you must configure CloudWatch with Amazon Organizations to turn on trusted access, and, typically, you'll also register a delegated administrator account. Then, using the delegated administrator account, you can add more accounts in your organization, to set the scope for your network observability to include resources in those accounts. (You can also add multiple accounts with a management account, but it's a best practice in Organizations to use the delegated administrator account when you work with resources in a service. We provide steps that follow that guidance in the instructions here for Network Flow Monitor.)

Note that if you don’t need to monitor network flows for instances from multiple accounts, you can use Network Flow Monitor with a single account. The scope for Network Flow Monitor is automatically set to the Amazon account that you sign in with.

Use the guidance in the following sections to complete these steps.

Contents

Overview of steps for using multiple accounts in Network Flow Monitor

To get started with Network Flow Monitor, any account that has not used Network Flow Monitor before must initialize Network Flow Monitor. When you initialize Network Flow Monitor for an account, Network Flow Monitor adds the required service-linked role permissions and creates a scope of the account or accounts to be included in network observability. To work with multiple accounts in Network Flow Monitor, there are additional steps, to integrate with Amazon Organizations, and then add the accounts to work with.

In summary, you take the following steps:

1. Sign in to the Amazon Web Services Management Console as the management account, and then do the following:
   - Complete the required steps for integrating with Amazon Organizations in CloudWatch.
2. Sign in to the Amazon Web Services Management Console as the delegated administrator account, and then do the following:
   - Initialize Network Flow Monitor, including adding accounts to include in your scope.
   - Add the required permissions for accessing resources that are in other accounts from the console.

If you're setting up Network Flow Monitor to work with multiple accounts and you're not familiar with Amazon Organizations, review the following resources to learn about concepts such as the management account, trusted access, and the delegated administrator account, and to learn how to integrate Organizations with CloudWatch.

- Managing accounts in an organization with Amazon Organizations in the Amazon Organizations User Guide.
- Amazon CloudWatch and Amazon Organizations in the Amazon Organizations User Guide.

Follow the steps in the following sections for specific guidance in configuring Network Flow Monitor for multiple accounts.

Configure Amazon Organizations in CloudWatch

To configure Network Flow Monitor with Amazon Organizations, sign in to the management account, and turn on trusted access for CloudWatch. Then, register a delegated administrator account to use for initializing Network Flow Monitor and adding multiple accounts.

If you've already configured Organizations in CloudWatch to turn on trusted access for Organizations in CloudWatch and register a delegated administrator account, you don't need to configure anything more for Organizations that is specific to Network Flow Monitor. You can sign in with the delegated administrator account for CloudWatch, and then initialize Network Flow Monitor, including adding multiple accounts for your network observability scope.

If you haven't yet configured Organizations in CloudWatch, follow the steps here to turn on trusted access and register a delegated administrator account.

Turn on trusted access in CloudWatch

Before you can use Network Flow Monitor with more than one account in your organization, you must turn on trusted access for Amazon Organizations in Amazon CloudWatch. Use the following steps to turn on trusted access in the CloudWatch console.

To turn on trusted access

1. Sign in to the console with your organization's management account.
2. In the CloudWatch console, in the navigation pane, choose **Settings**.
3. Choose the **Organizations** tab.
4. In **Organizational Management Settings**, choose **Turn on**. The **Enable trusted access** page appears.
5. To review the role policy, choose **View permission** details to see the role policy.
6. Choose **Enable trusted access**.

Now, as CloudWatch discovers resources, it automatically updates information about accounts that you have permission to access the resources for in Network Flow Monitor.

Register a delegated administrator account

As a best practice with Amazon Organizations, the management account for your organization should register a member account as a delegated administrator account for CloudWatch. After you register a delegated administrator account in CloudWatch, members of your organization can sign in with the delegated administrator account to monitor the network performance for resources in multiple accounts in Network Flow Monitor.

Using the delegated administrator account, you can add multiple accounts for your network observability scope in Network Flow Monitor. Although the management account can also create a scope that includes multiple accounts, we recommend that you follow the best practices for Amazon Organizations and use a delegated administrator account for adding multiple accounts in Network Flow Monitor. For member accounts that are not the delegated administrator account, the scope is limited to the signed-in account, which is automatically set for the scope.

A delegated administrator account for Organizations is a member account that shares administrator access for service-managed permissions. The account that you choose to register as a delegated administrator account must be a member account in your organization. A delegated administrator account for your organization can be used outside of CloudWatch, so make sure that you understand this account type before following this procedure. For more information, see Amazon CloudWatch and Amazon Organizations in the Amazon Organizations User Guide.

To register a delegated administrator account

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Settings**.
3. Choose the **Organization** tab.
4. Choose **Register delegated administrator**.
5. In the **Register delegated administrator** window, in the **Delegated administrator account ID** field, enter the 12-digit Organization member account ID.
6. Choose **Register delegated administrator**. At the top of the page, a message appears indicating the account was registered successfully. The **Organization Settings** page appears. To see information about the delegated administrator account, hover over the number below **Delegated administrators**.

To remove or change the delegated administrator account, deregister the account first. For more information, see Deregistering a delegated administrator account.

Add multiple accounts to your scope

To add accounts to your Network Flow Monitor scope, sign in with the delegated administrator account. (You can add accounts to a scope if you're signed in with the management account, but it's a best practice in Amazon Organizations to use the delegated administrator account to work with resources.)

After you sign in with the delegated administrator account, initialize Network Flow Monitor to authorize the required service-linked role permissions, set the scope for your network observability by adding accounts, and create an initial topology for the accounts in your scope. The account that you sign in with—in this case, the delegated administrator account—is automatically included in your Network Flow Monitor scope. To add accounts to your scope so that you can monitor network flows for resources in multiple accounts, follow the steps here.

To add accounts to your scope

1. Sign in to the console with your organization’s management account.
2. In the navigation pane for the CloudWatch console, under **Network Monitoring**, choose **Flow monitors**.

3. Under **Getting started with Network Flow Monitor**, in Step 1, choose **Start initialization**.
4. On the **Network Flow Monitor** page, under **Add accounts**, choose **Add**. The account that you're signed in with is automatically included in the scope and already appears in the **Accounts in scope** table as **(this account)**.
5. On the **Add accounts** dialog page, optionally filter the accounts, and then select up to 99 additional accounts to add to your scope. The maximum number of accounts in a scope is 100.
6. Choose **Add**.
7. Choose **Initialize Network Flow Monitor**. Network Flow Monitor adds the required service-linked role permissions, creates a scope that includes all the accounts you specified, and then creates an initial topology of the resources in the accounts in your scope.

Set up permissions for multi-account resource access (console only)

If you plan to create monitors for network flows from the console, a specific policy is required for each member account in your scope. This policy enables you to view resources from other accounts when you add local and remote resources to a monitor.

For each of the account in your scope, create a role, **NetworkFlowMonitorAccountResourceAccess**, and attach the **AmazonEC2ReadOnlyAccess** policy. To see permission details for the policy, see AmazonEC2ReadOnlyAccess in the Amazon Managed Policy Reference Guide.

This policy is in addition to the policy that you must add to each instance so that the Network Flow Monitor agent can send performance metrics from the instance to the Network Flow Monitor ingestion backend server. For more information about requirements for agents, see Install Network Flow Monitor agents on instances.

The following procedure provides a summary of the steps to create the required role for accessing resources in your scope in the Network Flow Monitor console. For general guidance on how to create a role in IAM, see Create a role to give permissions to an IAM user in the Amazon Identity and Access Management User Guide.

To create a role for resource access in the Network Flow Monitor console

1. Sign in to the Amazon Web Services Management Console and open the IAM console.
2. In the navigation pane of the console, choose **Roles**, and then choose **Create role**.
3. Specify the **Amazon account** trusted entity. This trusted entity type enables principals in other Amazon accounts to assume the role and access resources in other accounts.

4. Choose **Next**.
5. In the list of Amazon managed policies, choose the **AmazonEC2ReadOnlyAccess** policy.
6. Choose **Next**.
7. For role name, enter **NetworkFlowMonitorAccountResourceAccess**.
8. Review the role, and then choose **Create role**.

Install agents on instances

To track network performance with Network Flow Monitor, you must initialize the service, but you must also install Network Flow Monitor agents on your workload's EC2 instances and add permissions for the agents to send networking performance metrics to Network Flow Monitor. After you install the agents, wait a short period of time (about 20 minutes), for data to begin being sent to the Network Flow Monitor backend. Then, you can view network performance metrics, on the **Workload insights** tab, and also create monitors, to view detailed information.

For example, you can view the top contributor performance metrics for data transferred and retransmission timeouts, for network flows between your local and remote resources, collected by Network Flow Monitor agents. By viewing and analyzing these metrics, you can choose specific flows that you want to see more details for and track more closely with a monitor. By creating a monitor for specific flows, you can see detailed information about them, including metrics, sorted by the top contributors for each metric type and topologies for each network flow.

With a monitor, Network Flow Monitor also provides a network health indicator (NHI), which you can use to see if there have been Amazon network impairments for network flows that you're tracking in the monitor, during a time period that you've selected. That information can help you decide where to focus your network troubleshooting efforts.

For more information, and instructions for how to install agents, see [Install Network Flow Monitor agents on instances](#).

Monitor and analyze network flows in Network Flow Monitor

With Network Flow Monitor, you can learn about network flows and performance for traffic that you're monitoring with your scope. You can get workload insights about top contributors, by each metric type, and then create monitors to explore network performance in detail, over different time frames, for the network flows that you select.

After you initialize Network Flow Monitor and install agents on your instances, you can view performance metrics for the network flows from those instances. Review the sections here to learn more about how to use Network Flow Monitor to evaluate network performance in Network Flow Monitor, create monitors for more in-depth information, and learn about the specific metrics provided by Network Flow Monitor.

The steps provided in these sections use the Amazon Web Services Management Console. You can also use Network Flow Monitor API operations with the Amazon Command Line Interface (Amazon CLI) or Amazon SDKs to create and configure a monitor.

For detailed information about working with Network Flow Monitor API operations, see the following resources:

- If you plan to work with Network Flow Monitor with the CLI, see Examples of using the CLI with Network Flow Monitor.
- For detailed information about working with Network Flow Monitor API operations, see the Network Flow Monitor API Reference Guide.

Contents

Evaluate network flows with workload insights

Network Flow Monitor provides workload insights for the network flows for top contributors in the scope (account) that you enable monitoring for. Top contributors are network flows that have the highest values for each network performance metric. Top contributors are determined for workload insights, which includes all the network flows that you're receiving performance information about, as well as for the network flows you specify for each monitor that you create.

For example, Network Flow Monitor determines the network flows that have the highest values for retransmissions, retransmission timeouts, and data transferred. These network flows are the *top contributors* for each metric type. On the **Workload insights** page in the Network Flow Monitor console, Network Flow Monitor displays the top contributor network performance statistics for the network flows between all the resources in your monitoring scope where you've deployed agents.

Note that if you have multiple accounts in your scope that each have resources that you want to monitor network flows between, you must configure Amazon Organizations integration with CloudWatch before you can view performance metrics gathered by the Network Flow Monitor agents that you've installed. When you configure Organizations with Network Flow Monitor, you can add accounts to the scope for your Network Flow Monitor coverage.

Performance metrics on **Workload insights** are shown in separate tables for each metric type: Retransmissions, retransmission timeouts, and data transferred. The data provided is for the top contributors for each type. Note that after you first install Network Flow Monitor agents, there is a waiting period (about 20 minutes) before you can view performance metrics, while agents gather and send data to the Network Flow Monitor backend.

As you review performance metrics, when you see specific resources or network flows that you want to explore more details for, you can create a monitor that includes just those flows.

Create and work with monitors in Network Flow Monitor

You create a monitor to see details about the network performance for one or several network flows for a workload. For each monitor, Network Flow Monitor publishes end-to-end performance metrics and a network health indicator (NHI), and generates topologies of individual network flows. After you create a monitor, you can view information provided by the monitor in the console on the **Monitors** page.

After you create a monitor, you can edit the monitor to make changes or delete the monitor at any time.

The following sections includes procedures for creating, editing, and deleting monitors in the Network Flow Monitor console.

Contents

Create a monitor in Network Flow Monitor

You create a monitor by providing the local and remote resources to monitor flows between. You can select network flows on the **Workload insights** page, and then choose to create a monitor.

When you create a monitor in this way, the **Create monitor** wizard pre-populates local and remote resources for you. Alternatively, you can choose **Create monitor**, and then select local and remote resources for the monitor from drop-down menus.

By reviewing the top contributors on **Workload insights** on an ongoing basis, you can regularly evaluate if you have the monitors that you need, or if creating new monitors would be helpful.

> **Important**
>
> These steps are designed to be completed all at once. You won't be able to save any in-process work to continue later.

To create a monitor using the console

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the left navigation pane, under **Network Monitoring**, choose **Flow monitors**.
3. Choose **Create monitor**.
4. For **Monitor name**, enter the name you want to use for this monitor in Network Flow Monitor.
5. Select the local resources (one or more) for the network flows that you want to monitor. If your scope includes multiple accounts, choose an account for the resources that you want to specify for your local resources. Then, choose Subnet, VPC, or Availability Zone, and then choose a resource.

   If you started by choosing specific network flows that you wanted to monitor on the **Workload insights** page before starting the **Create monitor** process, network flows are pre-populated for you. If you like, modify your original choices by selecting resources to add or remove in this step.
6. Select the remote resources for the network flows that you want to monitor. To monitor all network flows that originate with the local resources that you chose, select **Everywhere**.

   If you want to select specific resources for your remote resources and your scope includes multiple accounts, choose an account for the resources that you want to specify. Then, choose Subnet, VPC, or Availability Zone, and then choose a resource.
7. Optionally, you can add a tag for your monitor.
8. When you're finished adding remote and local resources, choose **Next** to review and confirm the network flows to monitor.

9. Choose **Create monitor**.

After you create a monitor, you can edit the monitor at any time, to add or remove network flows, or add or edit tags. You can also delete the monitor. Select a monitor, and then choose **Edit**. Note that you can’t change the name of a monitor.

To view the Network Flow Monitor dashboard

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Network monitoring**, then **Flow monitors**.

   The **Monitors** tab displays a list of the monitors that you have created.

To see more information about a specific monitor, choose a monitor.

Edit a monitor in Network Flow Monitor

You can edit a monitor at any time, to add or remove network flows, or add or edit tags.

Note that you can't change the name of a monitor after you create it.

> **Important**
>
> These steps are designed to be completed all at once. You won't be able to save any in-process work to continue later.

To edit a monitor using the console

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the left navigation pane, under **Network Monitoring**, choose **Flow monitors**.
3. On the **Monitors** tab, select a monitor, and then choose **Edit**.
4. Select the local or remote resources that you want add or remove for the monitor. If you have multiple accounts in your scope, specify the account where the resources are located, and then choose resources.
5. Optionally, you can add or remove a tag for your monitor.
6. When you're finished updating the monitor, choose **Next** to review and confirm the network flows to monitor.

7. Choose **Save monitor**.

Delete a monitor in Network Flow Monitor

To delete a monitor in Network Flow Monitor, follow the steps here.

To delete a monitor

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the left navigation pane, under **Network Monitoring**, choose **Flow monitors**.
3. Select a monitor, and then choose **Delete**.
4. In the dialog that appears, enter confirmation text, and then choose **Delete**.

Monitor and analyze network flows using Network Flow Monitor performance metrics

Network Flow Monitor data and graphs help you to visualize and track network issues. You can create monitors to see detailed information about specific network segments for your Amazon workloads, including a view of the topology for individual network flows. After you create one or more monitors in Network Flow Monitor, you can observe performance and metrics, and explore historical data, to find anomalies.

To see the information provided by a monitor, on the **Monitors** tab, choose a monitor in the **Monitors** table. Then, choose one of the following tabs for more information: **Overview**, **Historical explorer**, or **Monitor details**.

Overview tab

On the **Overview** tab, you can review the following, for time periods that you specify. To see a broader or narrower range of historical information, including the NHI and traffic summary data, adjust the time period selection at the top of the page.

Metrics are shown only for traffic flows that Network Flow Monitor classifies into designated destination categories. Metrics are displayed for flows between AZs (`INTER_AZ`), within AZs (`INTRA_AZ`), between VPCs (`INTER_VPC`), toward Amazon S3 buckets (`AMAZON_S3`), and toward Amazon DynamoDB (`AMAZON_DYNAMODB`). Flows are not classified into a destination category for scenarios such as the following: when traffic goes to another Region or to the internet, or when traffic goes through a shared resource in another account.

- **Network health indicator (NHI):** NHI alerts you to whether there there were Amazon network issues for one or more of the network flows tracked by your monitor, during the time frame that you've selected for viewing performance metrics. NHI is a binary value, that is, 1 or 0, which is shown in the console as **Degraded** or **Healthy**.
  - NHI is shown as **Degraded** if there were issues with the portion of the Amazon network that any network flow in the monitor traversed, at any time during the time frame that you select.
  - Otherwise, NHI is shown as **Healthy**.

  If the NHI is **Degraded**, you can view the **Network health indicator** bar graph for more information. The graph shows you when, during the selected time frame, there were Amazon network issues for the network flows tracked by your monitor.
- **Traffic summary:** Observe overall metrics for the flows tracked by this monitor, for the time period that you've selected. You can see average round-trip time, sums (totals) of transmission timeouts and retransmissions, and the average amount of data transferred for the flows in the monitor. Be aware that RTT data can be sparse because RTT is not always calculated.

Historical explorer tab

On the **Historical explorer** tab, you can dive deep into information about specific flows. You can review metrics and topologies for top contributor network flows for specified time frames. In the tables of metrics, you can filter the data by different categories of flows, such as flows between Availability Zones (`INTER-AZ`).

- **Metrics:** View detailed information for the top contributors for each metric type that Network Flow Monitor aggregates data for. Separate tables of top contributors are provided for retransmission timeouts, retransmissions, round-trip time, and data transferred.
- **Topologies:** To get an idea about where anomalies are occurring, you can view the path or *topology*of a network flow. When you choose a specific metric in a metrics table, the topology for that flow is displayed below the table.

Monitor details tab

On the **Monitor details** tab, you can see details about the monitor, including the monitor state, the ARN, when it was created and last updated, and the flows that are included.

As part of your regular use of Network Flow Monitor, we recommend that you periodically review the data on the **Workload insights** page to determine if there are new flows that show metrics anomalies that you want to track more closely over time. When you see a set of flows on the **Workload insights** page that you want to see details about, select the flows and create a monitor for them.

Delete scope for Network Flow Monitor

If you decide that you no longer want to monitor network flows using Network Flow Monitor, you can delete your Network Flow Monitor scope. When you delete your scope, you can no longer see network performance information.

Before you can delete your scope, you must delete all monitors. Be aware that it can take about 15 minutes to complete removing monitors after you request to delete them. For more information, see Delete a monitor in Network Flow Monitor.

To delete your scope

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the left navigation pane, under **Network Monitoring**, choose **Flow monitors**.
3. On the **Settings** tab, choose **Delete scope**.
4. In the dialog box, enter confirmation text, and then choose **Delete scope**.

View Network Flow Monitor metrics in CloudWatch

Network Flow Monitor publishes the following network flow performance metrics to your account: round-trip time, TCP retransmissions, TCP retransmission timeouts, data transferred, and network health indicator. You can view these metrics in CloudWatch Metrics in the Amazon CloudWatch console.

To find all metrics for your monitor, in the CloudWatch Metrics dashboard, see the custom namespace `AWS/NetworkFlowMonitor`. Metrics are aggregated for each monitor that is deployed and active.

Network Flow Monitor provides the following metrics. Be aware that RoundTripTime data can be sparse, as this metric is not always calculated.

Metric	Description
DataTransferred	The number of bytes transferred for all flows for a monitor.
Retransmissions	Total number of retransmissions for a monitor. Retransmissions occur when the sender needs to resend packets that have been either damaged or lost.
Timeouts	Total retransmission timeouts for a monitor. This occurs when the sender is missing too many acknowledgments, and therefore decides to take a time out and stop sending altogether.
RoundTripTime	Average round-trip time for network flows for a monitor. This metric, measured in microseco nds, is a measure of performance. It records the time it takes for traffic to be transmitted from a local resource to a remote IP address, and for the associated response to be received. The time is an average over the aggregation period. Data can be sparse, as this metric is not always calculated.
HealthIndicator	Network health indicator (NHI) for a monitor overall. Network health indicator (NHI) is a value that surfaces an Amazon network impairment. The NHI value is 1 (degraded) if there was an Amazon network issue during a specified time frame. It's set to 0 (healthy) if no Amazon network issues were detected. Observing the NHI can help you to prioritize troubleshooting for either your workload or the Amazon network.

Create alarms with Network Flow Monitor

You can create Amazon CloudWatch alarms based on Network Flow Monitor metrics, just as you can for other CloudWatch metrics.

For example, you can create an alarm based on the Network Flow Monitor metric `Retransmissions`, and configure it to send a notification when the metric is higher than a value that you choose. You configure alarms for Network Flow Monitor metrics following the same guidelines as for other CloudWatch metrics.

Following are example Network Flow Monitor metrics that you might choose to create an alarm for:

- **Retransmissions**
- **Timeouts**
- **RoundTripTime**

To see all the metrics available for Network Flow Monitor see [Create a CloudWatch alarm based on a static threshold](#).

The following procedure provides an example of setting an alarm on **Retransmissions** by navigating to the metric in the CloudWatch dashboard. Then, you follow the standard CloudWatch steps to create an alarm based on a threshold that you choose, and set up a notification or choose other options.

To create an alarm for Retransmissions in CloudWatch Metrics

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. Choose **Metrics**, and then choose **All metrics**.
3. Filter for Network Flow Monitor by choosing `AWS/NetworkFlowMonitor`.
4. Choose **MeasurementSource, MonitorName**.
5. In the list, select **Retransmissions**.
6. On the **GraphedMetrics** tab, under **Actions**, choose the bell icon to create an alarm based on a static threshold.

Now, follow the standard CloudWatch steps to choose options for the alarm. For example, you can choose to be notified by an Amazon SNS message if **Retransmissions** is below a specific threshold number. Alternatively, or in addition, you can add the alarm to a dashboard.

Keep in mind the following:

- Network Flow Monitor metrics are typically aggregated and sent to the Network Flow Monitor backend every 30 seconds, with a 5 second potential jitter (in other words, 25 to 35 seconds).
- When you create an alarm based on Network Flow Monitor metrics, make sure that you take into account the short delay before publication when you set an alarm's lookback period. We recommend that you configure **Evaluation Periods** with lookback period that is a minimum of 25 minutes.

For more information about options when you create a CloudWatch alarm, see Create a CloudWatch alarm based on a static threshold.

Amazon CloudTrail for Network Flow Monitor

Monitoring a service is an important part of maintaining reliability, availability, and performance. of Network Flow Monitor and your other Amazon solutions. *Amazon CloudTrail* captures API calls and related events made by or on behalf of your Amazon Web Services account and delivers the log files to an Amazon S3 bucket that you specify. You can identify which users and accounts called Amazon, the source IP address from which the calls were made, and when the calls occurred. For more information, see the Amazon CloudTrail User Guide.

For more information about Network Flow Monitor CloudTrail logging, see Network Flow Monitor in CloudTrail.

Troubleshoot issues in Network Flow Monitor

This section provides guidance for troubleshooting errors with Network Flow Monitor, including solving issues with installing agents.

Troubleshoot issues in EKS agents installation

When you try to upgrade the Amazon Network Flow Monitor Agent add-on for EKS from v1.0.0 to v1.0.1 in Amazon Web Services Management Console, you might receive the following error message:

"Service account `aws-network-flow-monitoring-agent-service-account` in pod identity configuration is not supported for addon `aws-network-flow-monitoring-agent`."

This error is returned because a resource was renamed. The EKS add-on v1.0.1 changes the service account name from `aws-network-flow-monitoring-agent-service-account` to `aws-network-flow-monitor-agent-service-account`.

Then, if **Not set** is not selected in the console, the pod identity association is not reset to the new resource name.

To fix this issue, do the following when you upgrade to the new version by using the console:

1. Under **Pod Identity IAM role for service account**, select **Not set**.
2. Select **New version (v1.0.1)**.
3. Select **Upgrade**.
4. Choose **Save changes**.

Data security and data protection in Network Flow Monitor

Cloud security at Amazon is the highest priority. As an Amazon customer, you benefit from data centers and network architectures that are built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between Amazon and you. The shared responsibility model describes this as security *of* the cloud and security *in* the cloud:

- **Security of the cloud** – Amazon is responsible for protecting the infrastructure that runs Amazon services in the Amazon Web Services Cloud. Amazon also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the Amazon Compliance Programs. To learn about the compliance programs that apply to Network Flow Monitor, see Amazon Services in Scope by Compliance Program.
- **Security in the cloud** – Your responsibility is determined by the Amazon service that you use. You are also responsible for other factors including the sensitivity of your data, your company's requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using Network Flow Monitor. The following topics show you how to configure Network Flow

Monitor to meet your security and compliance objectives. You also learn how to use other Amazon services that help you to monitor and secure your Network Flow Monitor resources.

Topics

Data protection in Network Flow Monitor

The Amazon shared responsibility model applies to data protection in Network Flow Monitor. As described in this model, Amazon is responsible for protecting the global infrastructure that runs all of the Amazon Web Services Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. You are also responsible for the security configuration and management tasks for the Amazon Web Services services that you use. For more information about data privacy, see the Data Privacy FAQ.

For data protection purposes, we recommend that you protect Amazon Web Services account credentials and set up individual users with Amazon IAM Identity Center or Amazon Identity and Access Management (IAM). That way, each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with Amazon resources. We require TLS 1.2 and recommend TLS 1.3.
- Set up API and user activity logging with Amazon CloudTrail. For information about using CloudTrail trails to capture Amazon activities, see Working with CloudTrail trails in the *Amazon CloudTrail User Guide*.
- Use Amazon encryption solutions, along with all default security controls within Amazon Web Services services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing sensitive data that is stored in Amazon S3.
- If you require FIPS 140-3 validated cryptographic modules when accessing Amazon through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see Federal Information Processing Standard (FIPS) 140-3.

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form text fields such as a **Name** field. This includes when you work with Network Flow Monitor or other Amazon Web Services services using the console, API, Amazon CLI, or Amazon SDKs. Any data that you enter into tags or free-form text fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

Infrastructure Security in Network Flow Monitor

As a managed service, Network Flow Monitor is protected by the Amazon global network security procedures that are described in the [Amazon Web Services: Overview of Security Processes](#) whitepaper.

You use Amazon published API calls to access Network Flow Monitor through the network. Clients must support Transport Layer Security (TLS) 1.0 or later. We recommend TLS 1.2 or later. Clients must also support cipher suites with perfect forward secrecy (PFS) such as DHE (Ephemeral Diffie-Hellman) or ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the [Amazon Security Token Service](#) (Amazon STS) to generate temporary security credentials to sign requests.

Identity and Access Management for Network Flow Monitor

Amazon Identity and Access Management (IAM) is an Amazon Web Services service that helps an administrator securely control access to Amazon resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use Network Flow Monitor resources. IAM is an Amazon Web Services service that you can use with no additional charge.

Contents

How Network Flow Monitor works with IAM

Before you use IAM to manage access to Network Flow Monitor, learn what IAM features are available to use with Network Flow Monitor.

To see tables showing a similar high-level view of how Amazon services work with most IAM features, see Amazon services that work with IAM in the *IAM User Guide*.

IAM features you can use with Network Flow Monitor

IAM feature	Network Flow Monitor support
Identity-based policies	Yes
Resource-based policies	No
Policy actions	Yes
Policy resources	Yes
Policy condition keys (service-specific)	Yes
ACLs	No
ABAC (tags in policies)	Yes
Temporary credentials	Yes
Principal permissions	Yes
Service roles	No
Service-linked roles	Yes

Identity-based policies for Network Flow Monitor

Supports identity-based policies: Yes

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Define custom IAM permissions with customer managed policies in the *IAM User Guide*.

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. You can't specify the principal in an identity-based policy because it applies to the user or role to which it is attached. To learn about all of the elements that you can use in a JSON policy, see [IAM JSON policy elements reference](#) in the *IAM User Guide*.

Resource-based policies within Network Flow Monitor

Supports resource-based policies: No

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM role trust policies and Amazon S3 bucket policies. In services that support resource-based policies, service administrators can use them to control access to a specific resource.

Policy actions for Network Flow Monitor

Supports policy actions: Yes

Administrators can use Amazon JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Action` element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Policy actions usually have the same name as the associated Amazon API operation. There are some exceptions, such as *permission-only actions* that don't have a matching API operation. There are also some operations that require multiple actions in a policy. These additional actions are called *dependent actions*.

Include actions in a policy to grant permissions to perform the associated operation.

To see a list of Network Flow Monitor actions, see [Actions defined by Network Flow Monitor](#) in the *Service Authorization Reference*.

Policy actions in Network Flow Monitor use the following prefix before the action:

```
networkflowmonitor
```

To specify multiple actions in a single statement, separate them with commas.

```
"Action": [
      "networkflowmonitor:action1",
```

```
        "networkflowmonitor:action2"
           ]
```

You can specify multiple actions using wildcards (*). For example, to specify all actions that begin with the word `Describe`, include the following action:

```
"Action": "networkflowmonitor:Describe*"
```

Policy resources for Network Flow Monitor

Supports policy resources: Yes

In the *Service Authorization Reference*, you can see the following information related to Network Flow Monitor:

- To see a list of Network Flow Monitor resource types and their ARNs, see Resources defined by Network Flow Monitor.
- To learn the actions that you can specify with the ARN of each resource, see Actions defined by Network Flow Monitor.

Administrators can use Amazon JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Resource` JSON policy element specifies the object or objects to which the action applies. Statements must include either a `Resource` or a `NotResource` element. As a best practice, specify a resource using its Amazon Resource Name (ARN). You can do this for actions that support a specific resource type, known as *resource-level permissions*.

For actions that don't support resource-level permissions, such as listing operations, use a wildcard (*) to indicate that the statement applies to all resources.

```
"Resource": "*"
```

Policy condition keys for Network Flow Monitor

Supports service-specific policy condition keys: Yes

Administrators can use Amazon JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The Condition element (or Condition *block*) lets you specify conditions in which a statement is in effect. The Condition element is optional. You can create conditional expressions that use condition operators, such as equals or less than, to match the condition in the policy with values in the request.

If you specify multiple Condition elements in a statement, or multiple keys in a single Condition element, Amazon evaluates them using a logical AND operation. If you specify multiple values for a single condition key, Amazon evaluates the condition using a logical OR operation. All of the conditions must be met before the statement's permissions are granted.

You can also use placeholder variables when you specify conditions. For example, you can grant an IAM user permission to access a resource only if it is tagged with their IAM user name. For more information, see IAM policy elements: variables and tags in the *IAM User Guide*.

Amazon supports global condition keys and service-specific condition keys. To see all Amazon global condition keys, see Amazon global condition context keys in the *IAM User Guide*.

To see a list of Network Flow Monitor condition keys, see Condition keys for Network Flow Monitor in the *Service Authorization Reference*. To learn with which actions and resources you can use a condition key, see Actions defined by Network Flow Monitor.

ACLs in Network Flow Monitor

Supports ACLs: No

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

ABAC with Network Flow Monitor

Supports ABAC (tags in policies): Yes

Network Flow Monitor has *partial* support for tags in policies. It supports tagging for one resource, monitors.

To use tags with Network Flow Monitor, use the Amazon Command Line Interface or an Amazon SDK. Tagging for Network Flow Monitor is not supported with the Amazon Web Services Management Console.

To learn more about using tags in policies in general, review the following information.

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes. In Amazon, these attributes are called *tags*. You can attach tags to IAM entities (users or roles) and to many Amazon resources. Tagging entities and resources is the first step of ABAC. Then you design ABAC policies to allow operations when the principal's tag matches the tag on the resource that they are trying to access.

ABAC is helpful in environments that are growing rapidly and helps with situations where policy management becomes cumbersome.

To control access based on tags, you provide tag information in the condition element of a policy using the `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys.

If a service supports all three condition keys for every resource type, then the value is **Yes** for the service. If a service supports all three condition keys for only some resource types, then the value is **Partial**.

For more information about ABAC, see Define permissions with ABAC authorization in the *IAM User Guide*. To view a tutorial with steps for setting up ABAC, see Use attribute-based access control (ABAC) in the *IAM User Guide*.

Using temporary credentials with Network Flow Monitor

Supports temporary credentials: Yes

Some Amazon Web Services services don't work when you sign in using temporary credentials. For additional information, including which Amazon Web Services services work with temporary credentials, see Amazon Web Services services that work with IAM in the *IAM User Guide*.

You are using temporary credentials if you sign in to the Amazon Web Services Management Console using any method except a user name and password. For example, when you access Amazon using your company's single sign-on (SSO) link, that process automatically creates temporary credentials. You also automatically create temporary credentials when you sign in to the console as a user and then switch roles. For more information about switching roles, see Switch from a user to an IAM role (console) in the *IAM User Guide*.

You can manually create temporary credentials using the Amazon CLI or Amazon API. You can then use those temporary credentials to access Amazon. Amazon recommends that you dynamically generate temporary credentials instead of using long-term access keys. For more information, see Temporary security credentials in IAM.

Cross-service principal permissions for Network Flow Monitor

Supports forward access sessions (FAS): Yes

When you use an IAM user or role to perform actions in Amazon, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an Amazon Web Services service, combined with the requesting Amazon Web Services service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other Amazon Web Services services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see Forward access sessions.

Service roles for Network Flow Monitor

Supports service roles: No

A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Create a role to delegate permissions to an Amazon Web Services service in the *IAM User Guide*.

Service-linked role for Network Flow Monitor

Supports service-linked roles: Yes

A service-linked role is a type of service role that is linked to an Amazon Web Services service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your Amazon Web Services account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

For more information about the service-linked role for Network Flow Monitor, see Service-linked roles for Network Flow Monitor.

For details about creating or managing service-linked roles in general in Amazon, see Amazon services that work with IAM. Find a service in the table that includes a `Yes` in the **Service-linked role** column. Choose the **Yes** link to view the service-linked role documentation for that service.

Amazon managed policies for Network Flow Monitor

An Amazon managed policy is a standalone policy that is created and administered by Amazon. Amazon managed policies are designed to provide permissions for many common use cases so that you can start assigning permissions to users, groups, and roles.

Keep in mind that Amazon managed policies might not grant least-privilege permissions for your specific use cases because they're available for all Amazon customers to use. We recommend that you reduce permissions further by defining customer managed policies that are specific to your use cases.

You cannot change the permissions defined in Amazon managed policies. If Amazon updates the permissions defined in an Amazon managed policy, the update affects all principal identities (users, groups, and roles) that the policy is attached to. Amazon is most likely to update an Amazon managed policy when a new Amazon Web Services service is launched or new API operations become available for existing services.

For more information, see Amazon managed policies in the *IAM User Guide*.

Amazon managed policy: CloudWatchNetworkFlowMonitorServiceRolePolicy

You can't attach `CloudWatchNetworkFlowMonitorServiceRolePolicy` to your IAM entities. This policy is attached to a service-linked role named **AWSServiceRoleForNetworkFlowMonitor**, which publishes network telemetry aggregation results, collected by Network Flow Monitor agents, to CloudWatch. It also allows the service to use Amazon Organizations to get information for multi-account scenarios.

To view the permissions for this policy, see CloudWatchNetworkFlowMonitorServiceRolePolicy in the *Amazon Managed Policy Reference*.

For more information, see Service-linked roles for Network Flow Monitor.

Amazon managed policy: CloudWatchNetworkFlowMonitorTopologyServiceRolePolicy

You can't attach `CloudWatchNetworkFlowMonitorTopologyServiceRolePolicy` to your IAM entities. This policy is attached to a service-linked role named **AWSServiceRoleForNetworkFlowMonitor_Topology**. Using these permissions, as well as internal meta data information gathering (for performance efficiencies), this service-linked role gathers meta data about resource network configurations, such as describing route tables and gateways, for resources that this service monitors network traffic for. This meta data enables Network Flow Monitor to generate topology snapshots of the resources. When there is network degradation, Network Flow Monitor uses the topologies to provide insights into the location of issues in the network and to help determine attribution for issues.

To view the permissions for this policy, see CloudWatchNetworkFlowMonitorTopologyServiceRolePolicy in the *Amazon Managed Policy Reference*.

For more information, see Service-linked roles for Network Flow Monitor.

Amazon managed policy: CloudWatchNetworkFlowMonitorAgentPublishPolicy

You can use this policy in IAM roles that are attached to Amazon EC2 and Amazon EKS instance resources to send telemetry reports (metrics) to a Network Flow Monitor endpoint.

To view the permissions for this policy, see CloudWatchNetworkFlowMonitorAgentPublishPolicy in the *Amazon Managed Policy Reference*.

Updates to the Network Flow Monitor service-linked roles

For updates to the Amazon managed policies for the Network Flow Monitor service-linked roles, see the Amazon managed policies updates table for CloudWatch. You can also subscribe to automatic RSS alerts on the CloudWatch Document history page.

Service-linked roles for Network Flow Monitor

Network Flow Monitor uses Amazon Identity and Access Management (IAM) service-linked roles. A service-linked role is a unique type of IAM role that is linked directly to Network Flow Monitor. The service-linked role is predefined by Network Flow Monitor and includes all the permissions that the service requires to call other Amazon services on your behalf.

Network Flow Monitor defines the permissions of the service-linked roles, and unless defined otherwise, only Network Flow Monitor can assume the roles. The defined permissions include the trust policies and the permissions policies, and the permissions policies cannot be attached to any other IAM entity.

You can delete the roles only after first deleting their related resources. This restriction protects your Network Flow Monitor resources because you can't inadvertently remove permissions to access the resources.

For information about other services that support service-linked roles, see Amazon services that work with IAM and look for the services that have **Yes** in the **Service-linked role** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

Service-linked role permissions for Network Flow Monitor

Network Flow Monitor uses the following service-linked roles:

- **AWSServiceRoleForNetworkFlowMonitor**
- **AWSServiceRoleForNetworkFlowMonitor_Topology**

Service-linked role permissions for AWSServiceRoleForNetworkFlowMonitor

Network Flow Monitor uses the service-linked role named **AWSServiceRoleForNetworkFlowMonitor**. This role allows Network Flow Monitor to publish CloudWatch aggregated telemetry metrics gathered for network traffic between instances, and between instances and Amazon locations. It also allows the service to use Amazon Organizations to get information for multi-account scenarios.

This service-linked role uses the managed policy `CloudWatchNetworkFlowMonitorServiceRolePolicy`.

To view the permissions for this policy, see CloudWatchNetworkFlowMonitorServiceRolePolicy in the *Amazon Managed Policy Reference*.

The **AWSServiceRoleForNetworkFlowMonitor** service-linked role trusts the following service to assume the role:

- `networkflowmonitor.amazonaws.com`

Service-linked role permissions for AWSServiceRoleForNetworkFlowMonitor_Topology

Network Flow Monitor uses the service-linked role named **AWSServiceRoleForNetworkFlowMonitor_Topology**. This role allows Network Flow Monitor to generate a topology snapshot of the resources that you use with Network Flow Monitor.

This service-linked role uses the managed policy `CloudWatchNetworkFlowMonitorTopologyServiceRolePolicy`.

To view the permissions for this policy, see CloudWatchNetworkFlowMonitorTopologyServiceRolePolicy in the *Amazon Managed Policy Reference*.

The **AWSServiceRoleForNetworkFlowMonitor_Topology** service-linked role trusts the following service to assume the role:

- `topology.networkflowmonitor.amazonaws.com`

Creating a service-linked role for Network Flow Monitor

You do not need to manually create the service-linked roles for Network Flow Monitor. The first time that you initialize Network Flow Monitor, Network Flow Monitor creates **AWSServiceRoleForNetworkFlowMonitor** and **AWSServiceRoleForNetworkFlowMonitor_Topology** for you.

For more information, see Creating a service-linked role in the *IAM User Guide*.

Editing a service-linked role for Network Flow Monitor

After Network Flow Monitor creates a service-linked role in your account, you cannot change the name of the role because various entities might reference the role. You can edit the description of the role using IAM. For more information, see Editing a service-linked role in the *IAM User Guide*.

Deleting a service-linked role for Network Flow Monitor

If you no longer need to use a feature or service that requires a service-linked role, we recommend that you delete the role. That way you don't have an unused entity that is not actively monitored or maintained. However, you must clean up the resources for the service-linked role before you can manually delete it.

> **Note**
>
> If the Network Flow Monitor service is using the role when you try to delete it, then the deletion might fail. If that happens, wait for a few minutes and then try again.

To manually delete the service-linked role using IAM

Use the IAM console, the Amazon CLI, or the Amazon API to delete the **AWSServiceRoleForNetworkFlowMonitor** or the **AWSServiceRoleForNetworkFlowMonitor_Topology** service-linked role. For more information, see Deleting a service-linked role in the *IAM User Guide*.

Updates to the Network Flow Monitor service-linked role

For updates to `CloudWatchNetworkFlowMonitorServiceRolePolicy` or `CloudWatchNetworkFlowMonitorTopologyServiceRolePolicy`, the Amazon managed policies for the Network Flow Monitor service-linked roles, see CloudWatch updates to Amazon managed policies. For automatic alerts about managed policy changes in CloudWatch, subscribe to the RSS feed on the CloudWatch Document history page.

Network Flow Monitor quotas

Network Flow Monitor has the following quotas.

Resource	Default quota	Adjustable?
Scopes per account per Amazon Web Services Region	1	No
Monitors per account per Amazon Web Services Region	50	Yes
Local resources per monitor	25	No
Remote resources per monitor	25	No

Using Internet Monitor

Internet Monitor provides visibility into how internet issues impact the performance and availability between your applications hosted on Amazon and your end users. It can reduce the time it takes for you to diagnose internet issues from days to minutes. Internet Monitor uses the connectivity data that Amazon captures from its global networking footprint to calculate a baseline of performance and availability for internet-facing traffic. This is the same data that Amazon uses to monitor internet uptime and availability. With those measurements as a baseline, Internet Monitor raises awareness for you when there are significant problems for your end users (clients) in the different geographic locations where your application runs.

In the Amazon CloudWatch console, you can see a global view of traffic patterns and health events, and easily drill down into information about events, at different geographic granularities (locations). You can clearly visualize impact, and pinpoint the client locations and networks (ASNs, typically internet service providers or ISPs) that are affected. If Internet Monitor determines that an internet availability or performance issue is caused by a specific ASN or by the Amazon network, it provides that information.

To get started, create a monitor that includes one or more resources, so Internet Monitor can create a traffic profile for your Amazon application. Then, view information in the Internet Monitor dashboard to visualize data and get insights and suggestions about your application's internet traffic.

For information about Regional support, pricing, how Internet Monitor works, and other overview content, see What is Internet Monitor?. To begin working with Internet Monitor, see Getting started with Internet Monitor using the console.

What is Internet Monitor?

With Internet Monitor, you can monitor your application's internet performance and availability, so that you can visualize data and get insights and suggestions about your Amazon application's internet traffic. You can also get suggestions for ways to reduce latency for your application, by using different Regions or Amazon services, like Amazon CloudFront.

Key features of Internet Monitor

- Internet Monitor suggests insights and recommendations that can help you improve your end users' experience. You can explore, in near real-time, how to improve the projected latency of your application by switching to use other services, or by rerouting traffic to your workload through different Amazon Web Services Regions.
- Internet Monitor stores internet measurements for pairs of your client locations and ASNs, or *city-networks*. Internet Monitor also creates aggregated CloudWatch metrics for traffic to your application, and to each Amazon Web Services Region and edge location. With the Internet Monitor dashboard, you can quickly identify what's impacting your application's performance and availability, so that you can track down and address issues.
- Internet Monitor also publishes internet measurements to CloudWatch Logs and CloudWatch Metrics, to support using CloudWatch tools to explore data for city-networks that are specific to your monitored application traffic. Optionally, you can also publish internet measurements to Amazon S3.
- Internet Monitor sends overall (global) health events to Amazon EventBridge so that you can set up notifications. (Local health events are not published to EventBridge.) If an issue is caused by the Amazon network, you also automatically receive an Amazon Health Dashboard notification with the steps that Amazon is taking to mitigate the problem.

How to use Internet Monitor

To use Internet Monitor, you create a *monitor* and associate your application's resources with it—VPCs, Network Load Balancers, CloudFront distributions, or WorkSpaces directories—to enable Internet Monitor to know where your application's internet-facing traffic is. Internet Monitor then publishes internet measurements from Amazon that are specific to the *city-networks*, that is, the client locations and ASNs (typically internet service providers or ISPs), where clients access your application. For more information, see How Internet Monitor works. To begin working with Internet Monitor, see Getting started with Internet Monitor using the console.

Contents

Supported Amazon Web Services Regions for Internet Monitor

The Amazon Web Services Regions and Amazon Local Zones where Amazon CloudWatch Internet Monitor is supported are listed in this section. For more information about Regions that Internet Monitor is supported in, including opt-in Regions, see Amazon CloudWatch Internet Monitor endpoints and quotas in the *Amazon Web Services General Reference*.

Note that Internet Monitor stores data for a monitor in only the Amazon Web Services Region in which you create the monitor, although a monitor can include resources in multiple Regions.

Region name (Opt-in support)	Region
Africa (Cape Town)	`af-south-1`
Asia Pacific (Hong Kong)	`ap-east-1`
Asia Pacific (Hyderabad)	`ap-south-2`
Asia Pacific (Jakarta)	`ap-southeast-3`

Region name (Opt-in support)	Region
Asia Pacific (Melbourne)	ap-southeast-4
Europe (Milan)	eu-south-1
Europe (Spain)	eu-south-2
Europe (Zurich)	eu-central-2
Middle East (Bahrain)	me-south-1
Middle East (UAE)	me-central-1

Region name (Default support)	Region
US East (Ohio)	us-east-2
US East (N. Virginia)	us-east-1
US West (N. California)	us-west-1
US West (Oregon)	us-west-2
Asia Pacific (Mumbai)	ap-south-1
Asia Pacific (Osaka)	ap-northeast-3
Asia Pacific (Seoul)	ap-northeast-2
Asia Pacific (Singapore)	ap-southeast-1
Asia Pacific (Sydney)	ap-southeast-2
Asia Pacific (Tokyo)	ap-northeast-1
Canada (Central)	ca-central-1
Europe (Frankfurt)	eu-central-1

Region name (Default support)	Region
Europe (Ireland)	eu-west-1
Europe (London)	eu-west-2
Europe (Paris)	eu-west-3
Europe (Stockholm)	eu-north-1
South America (São Paulo)	sa-east-1

For Local Zones support, you must enable the Local Zone and attach it to the VPC that you want to monitor internet traffic for. Internet Monitor does not support Local Zones for other resources types. The Local Zones that are supported are listed in the following table.

Local Zone	Parent Region	Type
us-east-1-dfw-2a	us-east-1	Availability Zone
us-east-1-mia-2a	us-east-1	Availability Zone
us-east-1-qro-1a	us-east-1	Frontier Zone
us-east-1-lim-1a	us-east-1	Frontier Zone
us-east-1-atl-2a	us-east-1	Availability Zone
us-east-1-bue-1a	us-east-1	Frontier Zone
us-east-1-mci-1a	us-east-1	Frontier Zone
us-west-2-lax-1a	us-west-2	Availability Zone
us-west-2-lax-1b	us-west-2	Availability Zone
af-south-1-los-1a	af-south-1	Frontier Zone

Components and terms for Internet Monitor

Internet Monitor uses or references the following concepts.

Monitor

A monitor includes the resources for a single application that you want to view internet performance and availability measurements for, and that you want to get health event alerts about. When you create a monitor for an application, you add resources for the application to define the cities (locations) for Internet Monitor to monitor. Internet Monitor uses the traffic patterns from the application resources that you add so that it can publish internet performance and availability measurements specific to just the locations and ASNs (typically, internet service providers or ISPs) that communicate with your application. In other words, the resources that you add create a scope of the *city-networks* that you want Internet Monitor to monitor and that you want it to publish measurements for.

Resource added to monitor ("monitored resource")

A resource that you add to a monitor is a "monitored resource" in Internet Monitor. That is:

- Each VPC that you add in a Region is a monitored resource. When you add a VPC, Internet Monitor monitors the traffic for any internet-facing application in the VPC, for example, an application hosted on an Amazon EC2 instance, behind a Network Load Balancer, or an Amazon Fargate container.
- Each Network Load Balancer that you add in a Region is a monitored resource.
- Each WorkSpaces directory that you add in a Region is a monitored resource.
- Each CloudFront distribution that you add is a monitored resource.

Autonomous System Number (ASN)

In Internet Monitor, an ASN typically refers to an internet service provider (ISP), such as Verizon or Comcast. An ASN is a network provider that a client uses to access your internet application. An Autonomous System (AS) is a set of internet routable internet protocol (IP) prefixes that belong to a network or a collection of networks that are all managed, controlled, and supervised by one organization.

City-network (location and ASN)

A city-network is the location (such as a city) that clients access your application resources from and the ASN, typically an internet service provider (ISP), that clients access the resources through. To help control your bill, you can set a limit for the maximum number of city-networks for Internet Monitor to monitor for each monitor. You pay only for the actual number of city-networks that you monitor, up to the maximum number. For more information, see Choosing a city-network maximum limit.

Internet measurements

Internet Monitor also publishes internet measurements to log files in CloudWatch Logs every five minutes for the top 500 city-networks for your monitored application traffic.

These measurements quantify the performance score, availability score, bytes transferred (bytes in and bytes out), and round-trip time for your application's city-networks. These are measurements for the city-networks specific to your VPCs, Network Load Balancers, CloudFront distributions, or WorkSpaces directories. Optionally, you can choose to publish internet measurements and events for all monitored city-networks (up to the 500,000 city-networks service limit) to an Amazon S3 bucket.

Metrics

Internet Monitor generates aggregated metrics for CloudWatch metrics, for global traffic to your application and global traffic to each Amazon Web Services Region. For more information, see View Internet Monitor metrics or set alarms in CloudWatch Metrics.

Health event

Internet Monitor creates a health event to alert you to a specific problem that affects your application. Internet Monitor detects internet issues, such as increased network latency, across the world. It then uses its historical internet measurements from across the Amazon global infrastructure footprint to calculate the impact of current issues on your application, and creates health events. Internet Monitor, by default, creates health events based on both overall impact and local impact thresholds. To learn more about health events, see When Internet Monitor creates and resolves health events.

The default health event threshold, for both performance scores and availability scores, is 95%. If you like, you can specify your own custom thresholds for when Internet Monitor creates health events. For more information about configuring thresholds, see Change health event thresholds.

Each health event includes information about the impacted city-networks. You can view health events in the CloudWatch console, or by using an Amazon SDK or Amazon CLI with Internet Monitor API actions. Internet Monitor also sends Amazon EventBridge notifications for health events. For more information, see When Internet Monitor creates and resolves health events.

Internet event

Internet Monitor displays information about recent global health events, called internet events, on an internet weather map that is available to all Amazon customers. You don't need to create a monitor in Internet Monitor to view the internet weather map. Unlike health events, internet events are not specific to individual customers or their application traffic. For more information, see Global internet weather map in Internet Monitor.

Thresholds

Internet Monitor creates health events based on both overall thresholds and local thresholds. You can change the default thresholds and configure other options, such as turning off local thresholds. For more information about configuring thresholds, see Change health event thresholds.

Performance and availability scores

By analyzing the data that Amazon collects, Internet Monitor can detect when the performance and availability for your application has dropped, compared to estimated baselines that Internet Monitor calculates. To make it easier to see those drops, Internet Monitor reports the information to you as scores. A performance score represents the estimated percentage of traffic that is **not** seeing a performance drop. Similarly, an availability score represents the estimated percentage of traffic that is **not** seeing a availability drop. For more information, see How Amazon calculates performance and availability scores.

Bytes transferred and monitored bytes transferred

Bytes transferred is the total number of bytes of ingress and egress traffic between an application in Amazon and the city-network (that is, the location and the ASN, typically the internet service provider) where clients access an application. Monitored bytes transferred is a similar metric, but includes only bytes for monitored traffic.

Round-trip time

Round-trip time (RTT) is how long it takes for a request from a client user to return a response to the user. When RTT is aggregated across client locations (cities or other geographies), the value is weighted by how much of your application traffic is driven by each client location.

How Internet Monitor works

This section provides information about how Internet Monitor works. This includes descriptions of how Amazon collects the data that it uses to help detect connectivity issues across the internet, and how performance and availability scores are calculated.

Contents

How Internet Monitor focuses on just your application traffic footprint

Internet Monitor focuses monitoring on just the subset of the internet that's accessed by the users of your Amazon resources, instead of broadly monitoring your website from every Region in the world as other tools do. It's also a cost effective solution, affordable for large and small companies.

Internet Monitor uses the same powerful probes and issue-detection algorithms that Amazon takes advantage of internally and alerts you to connectivity issues that affect your application by creating health events in Internet Monitor. Internet Monitor then gives you access to the resulting performance and availability map, by overlaying the traffic profile that it creates from your active viewers, based on your application resources.

Using this information, Internet Monitor shows you just relevant events (that is, the events from places where you have active viewers), and just the impact those events have on your overall viewer volume. So, how much impact an event has, percentage-wise, is based on your total traffic worldwide.

Internet Monitor stores internet measurements for pairs of your client locations and ASNs, or *city-networks*. Internet Monitor also creates aggregated CloudWatch metrics for traffic to your application, and to each Amazon Web Services Region and edge location.

In addition, Internet Monitor publishes internet measurements to CloudWatch Logs internet every five minutes for the top 500 city-networks that send traffic to each monitor, to support using CloudWatch tools and other methods with your data. Optionally, you can choose to publish internet measurements for all monitored city-networks (up to the 500,000 city-networks service limit) to an Amazon S3 bucket. For more information, see Publish internet measurements to Amazon S3 in Internet Monitor.

The benefits of Internet Monitor include the following:

- Using Internet Monitor doesn't place additional load or cost on your application that's hosted on Amazon.
- You don't need to include performance measurement code in your client-side resources, or in your application.
- You can get visibility into performance and availability across the internet that your application is connected to, including "last mile" information.

Note that because Internet Monitor creates measurements based on your Amazon resources, Internet Monitor only creates events that are specific to your application traffic. Global internet issues in general are not reported. In addition, when the service location is an Amazon Web Services Region, the measurements and events emitted are designed to represent connectivity at a Regional level and don't accurately represent connectivity between an end user location and an Availability Zone.

How Amazon measures connectivity issues and calculates measurements

Internet Monitor uses internet connectivity data between different Amazon Web Services Regions and Amazon CloudFront points of presence (POPs) to different client locations through Autonomous System Numbers (ASNs), typically internet service providers (ISPs). This is the connectivity data that is used internally by Amazon operators, on a daily basis, to proactively detect connectivity issues across the global internet.

For every Amazon Web Services Region, we know which portions of the internet communicate with the Region and do the following:

- We actively monitor those portions of the internet, with a rolling 30-day window.
- We use both network and higher-level protocol probes, including both inbound and outbound probing.

Amazon has active and passive probes that measure the latency (performance) at the 90th percentile and reachability (availability) from every Amazon Web Services Region and from the

CloudFront service to the entire internet. Abnormal patterns in connectivity between a service and a customer location are monitored, and then reported as alerts to the customer.

See the following sections for details:

- Calculating availability and RTT
- Calculating performance and availability scores
- Calculating TTFB and RTT (latency)
- Regional and Availability Zone measurements and aggregation

Calculating availability and RTT

Round-trip time (RTT) is how long it takes for a request from the user to return a response to the user. When round-trip time is aggregated across end user locations, the value is weighted by the amount of your traffic that is driven by each end user location.

As an example, with two end user locations, one serving 90% of traffic with a 5 ms RTT, and the other serving 10% of traffic with a 10 ms RTT, the result is an aggregated RTT of 5.5 ms (which comes from 5 ms * 0.9 + 10 ms * 0.1).

Note that there are differences for resources about measuring last-mile latency. For Internet Monitor latency measurements, VPCs, Network Load Balancers, and WorkSpaces directories do not include last-mile latency.

Calculating performance and availability scores

Amazon has substantial historical data about internet performance and availability between Amazon services and different city-networks (locations and ASNs). By applying statistical analysis to the data, Internet Monitor can detect when the performance and availability for your application has dropped, compared to an estimated baseline that it has calculated. To make it easier to see those drops, that information is reported to you in the form of health scores: a performance score and an availability score.

Health scores are calculated at different granularities. At the finest granularity, we compute the health score for a geographic region, such as a city or a metro area, and an ASN (a *city-network*). We also roll up the individual health scores to overall health score numbers for an application in a monitor. If you view performance or availability scores without filtering for any specific geography or service provider, Internet Monitor provides overall health scores.

Overall health scores span your whole application for the specified time period. When the performance or availability score for your application's city-network pairs across your application reaches or drops below the corresponding health event threshold for performance or availability Internet Monitor triggers a health event. By default, the threshold is 95% for both overall performance and availability. Internet Monitor also creates health events based on local thresholds—if the option is enabled, as it is by default—based on values that you configure. To learn more about configuring health event thresholds, see Change health event thresholds.

When you explore information in the monitor and log files to investigate issues and learn more, you can filter by specific cities (locations), networks (ASNs or internet service providers), or both. So, you can use filters to see health scores for different cities, ASNs, or city-network pairs, depending on the filters that you choose.

- An *availability score* represents the estimated percentage of traffic that is **not** seeing an availability drop. Internet Monitor estimates the percentage of traffic experiencing a drop from the total traffic seen and availability metrics measurements. For example, an availability score of 99% for an end user and service location pair is equivalent to 1% of the traffic experiencing an availability drop for that pair.
- A *performance score* represents the percentage of traffic that is **not** seeing a performance drop. For example, a performance score of 99% for an end user and service location pair is equivalent to 1% of the traffic experiencing a performance drop for that pair.

Calculating TTFB and RTT (latency)

Time to first byte (TTFB) refers to the time between when a client makes a request and when it receives the first byte of information from the server. Amazon calculations for TTFB measure the time elapsed from Amazon EC2 or Amazon CloudFront to the Internet Monitor measurement node (including the last mile of the node). That is, Internet Monitor measures time from the user to the Amazon EC2 Region for TTFB for EC2, and from the user to CloudFront for TTFB for CloudFront.

For round-trip time (RTT), Internet Monitor includes the time from the city-network (that is, the client location and ASN, typically an internet service provider), as mapped by the public IP address, to the Amazon Web Services Region. This means that Internet Monitor does not have last mile visibility for users who access the internet from behind a gateway or VPN.

Note that there are differences for resources about measuring last-mile latency. For Internet Monitor latency measurements, VPCs, Network Load Balancers, and WorkSpaces directories do not include last-mile latency.

Internet Monitor includes average TTFB information in the **Traffic optimization suggestions** section of the **Traffic insights** tab on the CloudWatch dashboard, to help you evaluate options for different setups for your application that can improve performance.

Regional and Availability Zone measurements and aggregation

Although Internet Monitor aggregates measurements and shares impact at a Regional level, it calculates impact at an Availability Zone (AZ) level. This means that, if, for an event, only one AZ is impacted and most of your traffic flows through that AZ, you do see impact for your traffic. However, for the same event, if your application traffic does not flow through an impacted AZ, you do not see impact.

Note that this applies only to resources that aren't WorkSpaces directories. WorkSpaces directories are measured only on a Regional level.

Geolocation accuracy in Internet Monitor

For location information, Internet Monitor uses IP-geolocation data supplied by MaxMind. The accuracy of the location information in Internet Monitor measurements depends on the accuracy of MaxMind's data.

Be aware that `Metro` level measurements might not be accurate for locations outside of the United States.

When Internet Monitor creates and resolves health events

Internet Monitor creates and closes health events for the application traffic that you monitor based on the current thresholds that are set. Internet Monitor has a default threshold configuration, and you can also set your own configuration for thresholds. Internet Monitor determines the overall impact that connectivity issues are having on your application, and the impact on local areas where your application has clients, and creates health events when the thresholds are crossed.

Internet Monitor calculates the impact of connectivity issues on a client location based on the historical data about internet performance and availability for network traffic that's available to the service through Amazon. It applies the information relevant to your application, based on the geographic locations for ASNs and services where clients use your application: the city-network pairs that are affected. The locations are determined from the resources that you add to your monitor. Then Internet Monitor uses statistical analysis to detect when performance and availability has dropped, affecting the client experience for your application.

The performance and availability scores that Internet Monitor calculates are represented as the percentage of traffic that is **not** seeing a drop. Impact is the opposite of this: it's a representation of how much an issue is problematic for a customer's end users. So if there is a global availability drop of 93%, for example, the corresponding impact would be 7%.

When the performance or availability score for your application's city-network pairs globally reaches or drops below the corresponding health event threshold for performance or availability, this triggers Internet Monitor to generate a health event. By default, the threshold is 95% for both performance and availability. The values to meet, or drop below, the threshold are cumulative, so it could mean several smaller events combine to meet the threshold percentage, or that a single event meets or falls below the threshold level.

As long as performance or availability scores that triggered the event are at or below the corresponding health event threshold percentage for overall impact, the health event stays active. When the score or combined scores that triggered the event rise above the threshold, Internet Monitor resolves the health event.

Internet Monitor also creates health events based on local thresholds and the percentage of overall traffic that an issue has an impact on. You can configure options for local thresholds, or turn off local thresholds altogether.

The default health event threshold, for both performance scores and availability scores, is 95%. If you like, you can specify your own custom thresholds for when Internet Monitor creates health events. For more information about configuring thresholds, see [Change health event thresholds](#).

Health event report timing

Internet Monitor uses an aggregator to gather all signals about internet issues, to create health events in monitors within minutes.

When possible, Internet Monitor analyzes the origin of a health event, to determine whether it was caused by Amazon or an ASN. Health event analysis continues after an event is resolved. Internet Monitor can update events with new information for up to an hour.

How Internet Monitor works with IPv4 and IPv6 traffic

Internet Monitor measures health toward a network over only IPv4, and shows you health events, and availability and performance metrics, if you serve traffic to that network over any IP family (IPv4 or IPv6). If you serve traffic from a dual-stack resource, such as a dual-stack CloudFront distribution, Internet Monitor raises a health event and shows a drop in a performance score or availability score only if IPv4 traffic has the same issues for the resource as IPv6 traffic does.

Note that the Internet Monitor metrics for overall bytes in and bytes out accurately reflect all internet traffic (IPv4 and IPv6).

How Internet Monitor selects the subset of city-networks to include

When you set a maximum limit for the number of city-networks monitored by your monitor or choose a percentage of traffic to monitor, Internet Monitor chooses the city-networks to include (monitor) based on highest recent traffic volume.

For example, if you set a maximum city-networks limit of 100, Internet Monitor monitors (up to) 100 city-networks based on your application traffic during a recent one hour period. Specifically, Internet Monitor monitors the top 100 city-networks that have had the most traffic in the most recent one hour window *before* the latest one hour window.

To illustrate this, say that the current time is 2:30 PM. In this scenario, the traffic that you see in your monitor was captured between 1:00 PM and 2:00 PM, and the traffic volume measurement that Internet Monitor uses to determine the top 100 city-networks was captured between 12:00 PM and 1:00 PM.

How the global internet weather map is created (Frequently Asked Questions)

The Internet Monitor internet weather map is available on the Internet Monitor console to all authenticated Amazon customers. This section includes details about how the internet weather map is created and how to use it.

What is the Internet Monitor internet weather map?

The internet weather map provides a visual representation of internet issues across the world. It highlights impacted client locations, that is, cities plus ASN (typically internet service providers). The map shows a combination of availability and performance issues that have recently impacted clients' internet experience for top client locations and Amazon services globally.

Where does data for the map come from?

The data is based on a combination of active and passive probing of the internet. To learn more about how Internet Monitor measures data you can read the section [How Amazon measures connectivity issues](#).

How often is the map updated?

The internet weather map is updated every 15 minutes.

Which networks are tracked for outages?

Amazon tracks networks all around the world that represent important IP prefixes used by customers for making internet connections to Amazon. We scope outages to client locations that are top talkers for volume of traffic sent to and received from the Amazon network.

What determines whether an internet event is included on the map?

Here are some high level criteria that we use to determine whether an internet event is included on the internet weather map:

- Amazon detects that there is an availability or performance event.
- If the event is short lived, for example, it lasts less than 5 minutes, we ignore it.
- Then, if the event is in a client location that is classified as a top talker, it's considered an outage.

What thresholds are used for the internet weather map?

Thresholds for determining outages are not static for the internet weather map. Internet Monitor determines what constitutes an event based on detecting a deviation from expected values. You can learn more about how this works by reviewing how Internet Monitor determines when to create health events for monitors that you create with the service. When you create a monitor, Internet Monitor generates internet traffic health measurements that are specific to your own application traffic. Internet Monitor also alerts you to health events for issues that affect your application's internet traffic.

What can I do with this data?

The internet weather map provides a quick summary of key internet events that happened around the world in the last 24 hours. It helps you to get a sense of the internet monitoring experience, without needing to onboard your own internet traffic to Internet Monitor. To leverage the full potential of the internet monitoring capabilities of Amazon and to personalize it for your applications and services hosted on Amazon, you can create a monitor in Internet Monitor.

When you create a monitor, you enable Internet Monitor to identify the specific internet paths that affect your application clients, and you get access to features and capabilities that can help you improve your client experience. You'll also be proactively notified of new internet issues that specifically impact your application traffic and clients.

How can I get more details about events?

Click an outage on the map to see details that include when an event started and ended, the impacted city and ASN, and what type of issue it was (that is, a performance issue or an availability issue).

To get more detailed information about events, and to get custom measurements for your application traffic, create a monitor in Internet Monitor.

Internet Monitor example use cases

This section describes several specific examples of use cases for Internet Monitor, with links to blog posts with more details. These examples illustrate how you can use the capabilities of Internet Monitor to monitor your application health and improve latency to enhance your users' experience.

Set up alerts and decide on actions to take

You can use Internet Monitor to get insights about average internet performance metrics over time, and about health events by city-network (client location and ASN, typically an internet service provider). Using Internet Monitor, you can identify the events that are impacting end user experience for applications hosted on Amazon Virtual Private Clouds (VPCs), Network Load Balancers, Amazon WorkSpaces, or Amazon CloudFront.

After you create a monitor, you have several options for how to be alerted about Internet Monitor health events. These include notifications based on CloudWatch Alarms using event metrics or Amazon EventBridge rules to filter for health events. You can choose different options for notifications or actions based on alarms, including, for example, Amazon SMS notifications or updates to a CloudWatch log group.

To see an example with detailed guidance, see the following blog post: Introducing Internet Monitor.

Identify latency issues and improve TTFB to improve multiplayer gameplay experience

Use Internet Monitor to help you to quickly identify where game players in global cloud gaming apps are experiencing latency issues globally, and provide insights into improving performance. By identifying where the most players currently have the slowest time to first byte (TTFB), you know how to improve latency to make your biggest player base happier.

Now, when you're ready to deploy the next EC2 server for your game, choose the Amazon Web Services Region that Internet Monitor suggests will lower TTFB in the area with the high latency and large group of players.

For details about setting up and using Internet Monitor for this use case, see the following blog post: Using Internet Monitor for a Better Gaming Experience.

Identify potential performance and internet connection issues for users on Amazon WorkSpaces

Internet Monitor provides you with the IP prefixes and ASN (typically, the internet service provider or ISP) for your users, which can be helpful to diagnose performance and internet connection issues for users to their WorkSpaces. You can also use this data to view your fleet as a whole and monitor your WorkSpaces user connections.

For more information about how to use Internet Monitor for this use case, see the following blog post: Using Internet Monitor with Amazon WorkSpaces Personal.

Global internet weather map in Internet Monitor

Internet Monitor displays a global internet weather map that is available to all Amazon customers. To view the map, in the Amazon CloudWatch console, navigate to **Network Monitoring**, and then choose **Internet monitors**.

The internet weather map highlights internet events ("outages") all over the world that affect Amazon customers, with the specific cities and networks (ASNs, typically internet service providers) where there are issues with performance or availability. The map includes internet events from the past 24 hours.

You don't need to create a monitor in Internet Monitor to view the internet weather map. Unlike health events in Internet Monitor, internet events are not specific to individual customers or their application traffic.

On the internet weather map, you can choose an internet event to learn details about it. For an internet event, you can see the start time, end time (if the event is over), the current status (Active or Resolved), and the outage issue type (Availability or Performance). To learn more about how the internet weather map is created and what is included, see the global internet weather map FAQ.

To view and work with detailed information that is specific to your application traffic and client locations, you can create a monitor in Internet Monitor for your application. Then, you'll see performance and availability patterns and events, current and historical, as well as get health event alerts, tailored to just your application footprint and customers. The internet weather map gives you an overall view, while a specific monitor filters the information to just the measurements and details that are relevant to your application. With a monitor, you can also explore historical metrics and get recommendations for improving client experience for your application. To learn more, see Getting started with Internet Monitor using the console.

Internet Monitor cross-account observability

With Internet Monitor cross-account observability, you can monitor your applications that span multiple Amazon accounts within a single Amazon Web Services Region.

You can use Amazon CloudWatch Observability Access Manager to set up one or more of your Amazon accounts as a monitoring account. You'll provide the monitoring account with the ability to view data in your source account by creating a *sink* in your monitoring account. A sink is a resource that represents an attachment point in a monitoring account. For Internet Monitor, the resource attachment point is a monitor. You use the sink to create a link from your source account to your monitoring account. For more information, see CloudWatch cross-account observability.

Required resources

For proper functionality of CloudWatch Application Insights cross-account observability, ensure that the following telemetry types are shared through the CloudWatch Observability Access Manager.

- Monitors in Internet Monitor
- Metrics in Amazon CloudWatch
- Log groups in Amazon CloudWatch Logs

Pricing for Internet Monitor

With Internet Monitor, there are no upfront costs or long-term commitments. Pricing for Internet Monitor has two components: a per monitored resource fee and a per city-network fee. A *city-network* is the location where clients access your application resources from and the network (ASN, such as an internet service provider or ISP) that clients access the resources through. Note that you are also charged standard CloudWatch prices for logs and any additional metrics, dashboards, alarms, or insights that you create.

You choose a percentage of traffic to monitor when you create a monitor. To help control your bill, you can also set a limit for the maximum number of city-networks to monitor. You can update the percentage of traffic to monitor or the maximum city-networks limit at any time by editing your monitor. The first 100 city-networks (across all monitors per account) are included. After that, you only pay for the actual additional number of city-networks that you monitor, up to the maximum number.

You pay only the actual additional number of city-networks that you monitor, up to the maximum number, with no charge for the first 100 city-networks (across all monitors per account). A flat amount equivalent to the cost of 100 city-networks is deducted from your monthly bill.

For example, a large global company could choose to monitor 100% of its internet-facing traffic, and set a city-networks maximum of 50,000, for one monitor with one resource. Assuming the traffic reached 50,000 city-networks, that portion of its bill would be around 2,700 USD/month. For another company, in fewer geographic areas, with one monitor with one resource and 200 city-networks, this portion of the bill would be around 13 USD/month. For more information, see Choose a city-networks maximum limit.

You can try out different options with the pricing calculator. To explore pricing options, on the Pricing calculator for CloudWatch page, scroll down to Internet Monitor.

For more information about Internet Monitor and CloudWatch pricing, see the Amazon CloudWatch pricing page.

Getting started with Internet Monitor using the console

To help you get started with Internet Monitor, this chapter provides the steps for creating and configuring a *monitor*. You create a monitor in Internet Monitor for your application by naming it, and then adding Amazon resources that your application uses.

You create a monitor in Internet Monitor for your application by adding Amazon resources that it uses, and then setting several configuration options. The resources that you add, Amazon Virtual Private Cloud VPCs, Network Load Balancers (NLBs), CloudFront distributions, or WorkSpaces directories, provide the information for Internet Monitor to map internet traffic information for your application. After you create your monitor, wait 15-30 minutes to generate the traffic profile specific to where your application is used.

Then, use the Internet Monitor dashboard, or other tools, to visualize and explore performance and availability about your client usage. These tools provide insights for you using your application traffic's measurements gathered for you by the monitor.

The steps here walk you through setting up your monitor by using the console. To see examples of using the Amazon Command Line Interface with the Internet Monitor API actions, to create a monitor, view events, and so on, see Examples of using the CLI with Internet Monitor.

Tasks

Step 1: Create a monitor

To create a monitor using the console

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the left navigation pane, under **Network Monitoring**, choose **Internet monitors**.
3. Choose **Create monitor**.
4. For **Monitor name**, enter the name you want to use for this monitor in Internet Monitor.
5. Choose **Add resources**, and then select the resources to set the monitoring boundaries for Internet Monitor to use for this monitor.

   > **Note**
   >
   > Be aware of the following:
   >
   > - To generate meaningful output with Internet Monitor, VPCs that you add must be connected to the internet by having an Internet Gateway configured.
   > - You can add only one type of resource to a single monitor. For example, VPCs or CloudFront distributions or WorkSpaces directories, but not a combination of different types.

6. Leave the default percentage of traffic as 100%, or choose another percentage of your internet traffic to monitor.
7. Choose **Create monitor**.

Step 2: Configure the monitor

After you create a monitor, you can edit the monitor at any time, for example, to change the application traffic percentage, update the maximum city-networks limit or add or remove resources. To make updates in the Internet Monitor console, follow the procedure in this section. Note that you can't change the name of a monitor.

For more information about configuring a monitor, see Edit a monitor in Internet Monitor.

To configure a monitor using the console

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the left navigation pane, under **Network Monitoring**, choose **Internet monitors**.
3. Choose your monitor, and then choose the **Action** menu.
4. Choose **Update monitor**.
5. Make the desired updates. For example, to change the percentage of traffic to monitor, under **Application traffic to monitor**, select or enter a percentage.
6. Choose **Update**.

Step 3: View metrics and explore history

Visualize data about your internet traffic, from an overview perspective or by drilling down into details.

To visualize data and get insights for application traffic using the console

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the left navigation pane, under **Network Monitoring**, choose **Internet monitors**.
3. Choose a monitor to work with.
4. Choose from the following tabs:

   - **Overview** — Review a general summary of your monitor and your application traffic performance.
   - **Health events** — View current and historical health events that currently impact, or previously impacted, locations where clients access your application.

- **Analyze** — See information about top monitored traffic in client locations (by traffic volume), summarized in several customizable ways. Visualize metrics and historical trends for health scores and metrics.

In the next section, learn about how Internet Monitor provides suggestions for improving latency for your application traffic.

Step 4: Get suggestions to improve latency

Get suggestions for how to optimize latency, so that your clients experience the best internet performance for your application.

Internet Monitor evaluates your monitored application traffic, and then makes suggestion about whether you can reduce latency, for example, by changing the Amazon Web Services Regions that you've configured for your application.

For more information, see Get suggestions to optimize application performance in Internet Monitor (Optimize page).

To get suggestions for improving application latency using the console

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the left navigation pane, under **Network Monitoring**, choose **Internet monitors**.
3. Choose a monitor to work with.
4. Choose **Optimize**, and then view the top suggestions.

Step 5 (Optional): Delete the monitor

If you created a monitor as a test or if you're no longer using a monitor, you can delete it. Before you can delete a monitor, you must disable it.

To delete a monitor

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the left navigation pane, under **Network Monitoring**, choose **Internet monitors**.
3. Choose your monitor, and then choose the **Action** menu.
4. Choose **Disable**.

5. Choose the **Action** menu again, and then choose **Delete**.
6. Follow the guidance in the modal dialog to confirm deleting the monitor.

Configure a monitor using the console

This chapter includes procedures and recommendations for creating and configuring monitors in Internet Monitor.

The steps provided in these sections primarily use the Amazon Web Services Management Console. You can also use Internet Monitor API operations with the Amazon Command Line Interface (Amazon CLI) or Amazon SDKs to create and configure a monitor. For detailed information about working with Internet Monitor API operations, see the following resources:

- If you plan to work with Internet Monitor with the CLI, see Examples of using the CLI with Internet Monitor.
- For detailed information about working with Internet Monitor API operations, see the Internet Monitor API Reference.

Contents

Create a monitor in Internet Monitor using the console

You create a monitor in Internet Monitor to visualize and explore performance and availability data about your application's client traffic. You create a monitor by adding Amazon resources that your application uses, and then setting several configuration options. The resources that you add to the monitor provide the information — for example, through resource flow logs — for Internet Monitor to learn which internet traffic is specific to your Amazon application.

After you create your monitor, wait 15 to 30 minutes before reviewing the monitor dashboard. Internet Monitor needs a few minutes to generate a traffic profile for where your application is used by your end users, and then to begin publishing data for your traffic.

Typically, it's simplest to create one monitor in Internet Monitor for one application. Within the same monitor, you can search through and sort measurements and metrics by different locations and ASNs, or other information. You don't need to create separate monitors for applications in different areas.

The steps provided here walk you through setting up your monitor by using the console. To work with Internet Monitor API actions using the Amazon Command Line Interface, see Examples of using the CLI with Internet Monitor.

To create a monitor using the console

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the left navigation pane, under **Network Monitoring**, choose **Internet monitors**.
3. Choose **Create monitor**.
4. For **Monitor name**, enter the name that you want to use for this monitor.
5. Choose **Add resources**, and then select the resources that will determine the internet traffic profile for this monitor.

   > **Note**
   >
   > Be aware of the following:
   >
   > - To generate meaningful output with Internet Monitor, VPCs that you add must be connected to the internet by having an Internet Gateway configured.
   > - You can specify only one type of resource for each monitor.

6. Choose a percentage of your application's internet traffic to monitor.
7. Optionally, under **Advanced settings**, specify one or more of the following additional options.

   - **City-networks maximum** — The default city-networks maximum value is `500000`. If you like, you can lower this limit, to restrict the number of city-networks (locations and ASNs) that Internet Monitor will monitor traffic for. You can change the city-networks maximum at any time by editing your monitor. For more information, see Choose a city-networks maximum limit.

- **Amazon S3 bucket storage** — You can specify an Amazon S3 bucket name and custom prefix to publish internet measurements for your application's internet traffic to Amazon S3, for all monitored city-networks.

  Internet Monitor stores internet measurements for pairs of your client locations and ASNs, or *city-networks*. Internet Monitor also creates aggregated CloudWatch metrics for traffic to your application, and to each Amazon Web Services Region and edge location. In addition, Internet Monitor publishes internet measurements for your application traffic to CloudWatch Logs every five minutes, to support using CloudWatch tools and other methods with your data. If you choose to publish measurements to S3, measurements are still published to CloudWatch Logs. For more information, see Publish internet measurements to Amazon S3 in Internet Monitor.
- **Tags** — Add one or more tags for your monitor.

8. Choose **Create monitor**.

After you create a monitor, wait about 15-30 minutes for Internet Monitor to create a traffic profile and begin publishing your data. Then, you can view information about your application's internet traffic performance by navigating to the monitor dashboard in the console.

To view the Internet Monitor dashboard

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Network Monitoring**, then **Internet monitors**.
3. To see more information about a specific monitor, on the **Monitors** tab, choose a monitor.

Add resources to your monitor

When you create a monitor, you must associate your application's resources with it: Amazon Virtual Private Clouds (VPCs), Network Load Balancers, Amazon CloudFront distributions, Network Load Balancers (NLBs), or Amazon WorkSpaces directories. Then, Internet Monitor knows where your application's internet-facing traffic and clients are located, and it can create and maintain a traffic profile that determines the relevant measurements to publish for your monitor.

You can add the following types of resources to a monitor in Internet Monitor as *monitored resources*.

- **VPCs:** Each VPC that you add in a Region is a monitored resource. When you add a VPC, Internet Monitor monitors the traffic for any internet-facing application in the VPC, for example, an application hosted on an Amazon EC2 instance, behind a Network Load Balancer, or in an Amazon Fargate container.
- **Network Load Balancers:** Each NLB that you add is a monitored resource.
- **CloudFront distributions:** Each CloudFront distribution that you add is a monitored resource.
- **WorkSpaces directories:** Each WorkSpaces directory that you add in a Region is a monitored resource.

When you monitor traffic for VPCs, traffic for applications that are hosted on load balancers behind the VPC is monitored. You can choose to monitor traffic for individual Network Load Balancer load balancers instead of monitoring a VPC with multiple load balancers. This can be helpful, for example, if you need to understand and configure features for better performance or efficiencies at the load balancer level. Or, you might need compliance information at the Network Load Balancer level.

When you add resources to a monitor in Internet Monitor, be aware of the following:

- Internet Monitor doesn't support adding different types of resources together in one monitor.
- To generate meaningful output with Internet Monitor, VPCs that you add must be connected to the internet by having an Internet Gateway configured.
- Internet Monitor doesn't support adding different types of resources together in one monitor.
- There are Regional differences for opt-in Regions to keep in mind when you add VPCs or NLBs as resources. For more information, see [Supported Amazon Web Services Regions for Internet Monitor](#).
- In addition, there are differences for resources about measuring last-mile latency. For Internet Monitor latency measurements, VPCs, NLBs, and WorkSpaces directories do not include last-mile latency.

Choose a percentage of traffic to monitor for your application

The coverage that you choose for the percentage of application traffic to monitor determines how many city-networks (client locations and ASNs, typically internet service providers) for your application are monitored, up to an optional city-networks maximum limit that you can also set.

You can choose the percentage of traffic to monitor when you create a monitor, or, with an existing monitor, by choosing **Edit monitor** on any Internet Monitor dashboard page in the console.

If you choose to monitor less than 100% of your application traffic, you might have an observability gap in with your monitor. That's because if there are health events that Internet Monitor creates where you aren't monitoring traffic, you won't be aware of those issues. With a traffic percentage set to less than 100%, you might also have less coverage for the performance and availability score information about client access to your application.

The following sections describe options to explore traffic percentage settings and coverage, and to get an idea about the impact of increasing or decreasing coverage.

- Explore changing your application traffic percentage
- View the number of city-networks monitored at different traffic percentage settings

Explore changing your application traffic percentage

You can explore values that you might want to change your application traffic percentage to, by viewing the number of city-networks monitored when you change the percentage. The procedure in this section provides step-by-step information.

In the Internet Monitor console, you can try increasing or decreasing the application traffic percentage for your monitor, and view the estimated number of your city-networks that would be covered as a result. With this option, you can quickly see how changing your traffic percentage affects the number of city-monitors are monitored. This can help you to get a feel for what a good application traffic percentage to choose might be, for your application.

To explore monitoring coverage and update percentage of traffic monitored

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the left navigation pane, under **Network Monitoring**, choose **Internet monitors**.
3. In your list of monitors, choose a monitor.
4. On the **Configure** tab, in the **View and evaluate traffic coverage** section, you can evaluate the impact on the total number of city-networks that are monitored, depending on a traffic percentage that you choose. You can also update the percentage of traffic that you monitor or change the city-networks limit for your monitor.

    - **Explore traffic percentage options:** Under **Compare options for traffic coverage**, in the drop-down menu, choose one or more traffic percentages to graph and compare. For each traffic percentage that you choose, you can see the number of city-networks that will be monitored when you set that traffic percentage coverage.

To learn more, see View number of city-networks monitored at different percentages.

- **Change monitoring coverage:** Under **Explore other traffic coverage options**, choose **Update monitoring coverage**.

  In the **Explore and set traffic monitoring coverage** dialog, click the arrows to increase or decrease the percentage of traffic to monitor. By choosing 100% traffic, you can see how many city-networks are monitored with full coverage for monitoring your application.

  Note: To learn more about how the number of city-networks monitored (estimated here) might affect your costs, choose the link to the CloudWatch Pricing calculator, and then scroll down to Internet Monitor.

  To set a new percentage of traffic to monitor, choose **Update monitor coverage**. Or, to keep the current coverage level, choose **Cancel**.

View the number of city-networks monitored at different traffic percentage settings

You can view the number of city-networks that would be monitored for your application at different application traffic percentages. The procedure in this section provides step-by-step information.

In the Internet Monitor console, you can view graphs that show how coverage for your city-networks would change at different of application traffic percentages, over a time interval that you specify. This is a quick way to visualize and compare the monitoring coverage for your application at specific traffic percentages, all on one graph.

To view graphs of application traffic percentage and corresponding city-networks coverage

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the left navigation pane, under **Network Monitoring**, choose **Internet monitors**.
3. In your list of monitors, choose a monitor.
4. Choose the **Configure** page, and scroll down to **Traffic coverage**.
5. Under **Compare options for traffic coverage**, in the drop-down list, select one or more percentages. You can choose one or more application traffic percentages, and the graph of **Total monitored city-networks** is updated to display the monitoring coverage Internet

Monitor provides for that traffic percentage. By choosing **City-networks at 100% traffic**, you can see how many city-networks are monitored with full coverage for monitoring your application.

Keep in mind the following:

- Traffic coverage is computed based on the number of city-networks in the previous hour of your application traffic. This means that, after you choose a specific percentage of traffic to monitor, fewer city-networks might be monitored for your application than is shown here in a traffic coverage comparison graph.
- To make sure that all your application traffic is monitored, set `TrafficPercentageToMonitor` to 100 and don't set `MaxCityNetworksToMonitor`. Alternatively, you can set `MaxCityNetworksToMonitor` to 500,000, the upper limit in Internet Monitor.
- If you set a city-networks maximum limit, the total number of monitored city-networks never exceeds that limit, regardless of the application traffic percentage option that you select.
- You can learn more about how the number of city-networks monitored might affect your costs. On the Pricing calculator for CloudWatch page, scroll down to Internet Monitor.

To set a new percentage of traffic to monitor, under **Explore other traffic coverage options**, choose **Update monitoring coverage**. In the dialog, choose a percentage of traffic, and then choose **Update monitor coverage**.

Use a monitor in Internet Monitor

There are several ways to use an Internet Monitor monitor after you create it: for example, you can view information in the CloudWatch dashboard, get information by using the Amazon Command Line Interface, and set health alerts.

Your monitor provides information about your application and configuration preferences so that Internet Monitor can customize measurements and metrics to publish in events for you. Internet Monitor collects measurements from the global infrastructure footprint for Amazon. These measurements are a tremendous amount of network performance and availability information, from all over the world. By using information from the resources that you add for your application, Internet Monitor publishes performance and availability measurements for you that is scoped to the city-networks (that is, client locations and ASNs, typically internet service providers or ISPs) where your application is active. So, the measurements and metrics in the Internet Monitor dashboard and in CloudWatch Logs —about availability, performance, monitored bytes transferred, and round-trip time—are specific to your client locations and ASNs.

Internet Monitor also determines when there are anomalies in performance and availability. By default, Internet Monitor overlays your traffic with the availability and performance measurements that Amazon has collected for each source-destination pair in your client locations, to determine when there are notable drops in performance or availability. When there's significant degradation for your application's locations and scope, Internet Monitor generates a *health event*, and publishes information about the issue to your monitor.

After you create a monitor, you can use it to access or be alerted to the information that Internet Monitor provides, in the following ways:

- **Use the CloudWatch dashboard** to view and explore performance, availability, and health events; explore your application's historical data; and get insights into new ways to configure your application for better performance. To learn more, see the following:
  - Track real-time performance and availability in Internet Monitor (Overview page)
  - Analyze historical data in Internet Monitor (Analyze page)
  - Get suggestions to optimize application performance in Internet Monitor (Optimize page)
- **Configure health event thresholds** to change what triggers Internet Monitor to create a health event for your application. You can configure overall thresholds and local (city-network) thresholds. To learn more, see Change health event thresholds.
- **Use Amazon CLI commands** with Internet Monitor API actions to view traffic profile information, view measurements, list health events, and so on. To learn more, see Examples of using the CLI with Internet Monitor.
- **Use standard CloudWatch tools,** such as CloudWatch Contributor Insights, CloudWatch Metrics explorer, and CloudWatch Logs Insights to visualize the data in CloudWatch. To learn more, see Exploring your data with CloudWatch tools and the Internet Monitor query interface.
- **Use Athena with S3 logs** to access and analyze Internet Monitor internet measurements for your application, if you turned on publishing measurements to S3.
- **Create Amazon EventBridge notifications** to alert you when Internet Monitor determines there is a health event. To learn more, see Using Internet Monitor with Amazon EventBridge.
- **Receive an Amazon Health Dashboard notification** automatically, when Internet Monitor determines that an issue is caused by the Amazon network. The notification includes the steps that Amazon is taking to mitigate the problem.

Edit a monitor in Internet Monitor

Using the **Action** menu, you can edit a monitor in Amazon CloudWatch Internet Monitor after you create it. For example, you can edit a monitor to do the following:

- Change the percentage of application traffic to monitor
- Set or update the city-networks maximum limit
- Change health event thresholds for availability or performance scores
- Add or remove resources
- Enable or update publishing events to Amazon S3

Note that you can't change the name of a monitor after you create it.

To make changes to a monitor, use the following procedure.

To edit a monitor

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the left navigation pane, under **Network Monitoring**, choose **Internet monitors**.
3. Choose your monitor, and then choose the **Action** menu.
4. Choose **Update monitor**.
5. Make the desired updates. For example, to change the percentage of traffic to monitor, under **Application traffic to monitor**, select or enter a percentage.
6. Choose **Update**.

For more information about the options that you can update, see the following:

- To learn more about resources that you add in Internet Monitor, see Add resources to your monitor.
- To learn more about the application traffic percentage, see Choose a percentage of traffic to monitor for your application.
- To learn more about changing the threshold for health events, see Change health event thresholds.
- To learn more about the city-networks maximum limit, see Choose a city-networks maximum limit.

- To learn more about opting to publish events to S3, see Publish internet measurements to Amazon S3 in Internet Monitor.

Delete a monitor in Internet Monitor

Using the **Action** menu, you can delete a monitor in Amazon CloudWatch Internet Monitor. You first disable the monitor, and then delete it.

To delete a monitor

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the left navigation pane, under **Network monitoring**, choose **Internet Monitor**.
3. Choose your monitor, and then choose the **Action** menu.
4. Choose **Disable**.
5. Choose the **Action** menu again, and then choose **Delete**.

Advanced configuration options for a monitor

This section provides the steps for configuring advanced options for a monitor in Internet Monitor. These configuration options are optional but can be useful in some scenarios.

For example, you might choose to set a city-network maximum limit if traffic for the application that you monitor with Internet Monitor occasionally spikes, and you want to help make sure that your bill for Internet Monitor is predictable.

Or, you might want to set custom or local thresholds for health events, because you want to pay close attention to issues in specific geographies where you have a concentration of clients.

The topics in this section provide detailed descriptions of each feature, and list the steps to configure options for your needs.

Contents

Choose a city-networks maximum limit

In addition to setting a traffic percentage for your monitor in Internet Monitor, you can also set a maximum limit for the number of city-networks monitored. This section describes how the city-networks limit can help you manage billing costs, and provides information and an example to help you determine a limit to set, if you choose to set one.

Internet Monitor can monitor traffic for some or all of the locations where clients access your Amazon application resources. You can set a monitoring limit for the number of *city-networks*, that is, the client locations and the ASNs (typically internet service providers) that clients access your application through.

You choose a percentage of application traffic to monitor when you create your monitor. The default percentage is 100%. You can update the percentage at any time, by editing the monitor.

The maximum limit that you set for the number of city-networks helps to make sure that your bill is predictable. For more information, see Amazon CloudWatch Pricing. You can also learn how different values for the number of city-networks actually monitored can affect your bill by using the CloudWatch price calculator. To explore options, on the Pricing calculator for CloudWatch page, scroll down to Internet Monitor.

To update your monitor and change the maximum city-networks limit, see Edit a monitor in Internet Monitor.

How billing works with city-networks maximum limits

Setting a maximum limit for the number of city-networks monitored can help prevent unexpected costs in your bill. This is useful, for example, if your traffic patterns vary widely. Billing costs increase for each city-network that is monitored after the first 100 city-networks, which are included (across all monitors per account). If you set a city-networks maximum limit, it caps the number of city-networks that Internet Monitor monitors for your application, regardless of the percentage of traffic that you choose to monitor.

You only pay for the number of city-networks that are actually monitored. The city-network maximum limit that you choose lets you set a cap on the total that can be included when Internet Monitor monitors traffic with your monitor. You can change the maximum limit at any time by editing your monitor.

To explore options, on the Pricing calculator for CloudWatch page, scroll down to Internet Monitor. For more information on Internet Monitor pricing, see the Internet Monitor section on the Amazon CloudWatch Pricing page.

How to choose a city-networks maximum limit

Optionally, you can set a city-networks maximum limit. To help you decide on a maximum limit that you might want to select, consider how much traffic you want to monitor for your application. Be aware that if you choose 100% for the *traffic percentage to monitor* for your monitor, and then specify a city-networks maximum limit, depending on the limit that you choose, you might not monitor 100% of your application traffic. The city-networks maximum that you set takes precedence over the traffic percentage to monitor that you set.

To view how the percentage of traffic to monitor that you choose affects the number of city-monitors that are included for your application monitoring, which can help you decide whether to set a city-networks maximum limit, follow the steps in [View the number of city-networks monitored at different traffic percentage settings](#).

To explore your options in more detail, you can use Internet Monitor metrics, as described in the following examples. These examples show how to select a maximum city-networks limit that is best for you, depending on the breadth of application internet traffic coverage you want. Using the [queries for Internet Monitor metrics in CloudWatch Metrics](#) can help you understand more about your application internet traffic coverage.

Example of determining a city-networks maximum limit

As an example, say that you've set a monitoring maximum limit of 100 city-networks and that your application is accessed by clients across 2637 city-networks. In CloudWatch Metrics, you'd see the following Internet Monitor metrics returned:

```
CityNetworksMonitored 100
TrafficMonitoredPercent  12.5
CityNetworksFor90PercentTraffic  2143
CityNetworksFor100PercentTraffic  2637
```

From this example, you can see that you're currently monitoring 12.5% of your internet traffic, with the maximum limit set to 100 city-networks. If you want to monitor 90% of your traffic, the next metric provides information about that: `CityNetworksFor90PercentTraffic` indicates that you would need to monitor 2,143 city-networks for 90% coverage. To do that, you would update your monitor and set the maximum city-networks limit to 2,143.

Similarly, say you'd like to have 100% internet traffic monitoring for your application. The next metric, `CityNetworksFor100PercentTraffic`, indicates that to do this, you should update your monitor to set the maximum city-networks limit to 2,637.

If you now set the maximum to 5,000 city-networks, since that's greater than 2,637, you see the following metrics returned:

```
CityNetworksMonitored 2637
TrafficMonitoredPercent  100
CityNetworksFor90PercentTraffic  2143
CityNetworksFor100PercentTraffic  2637
```

From these metrics, you can see that with the higher limit, you monitor all 2,637 city-networks, which is 100% of your internet traffic.

Change health event thresholds for a monitor

Internet Monitor uses a default threshold to determine when to create a health event for your monitor. Optionally, you can change that default global threshold, to set another value. You can also set local threshold. This section describes how global and local thresholds work together, and provides steps for setting custom thresholds.

You can change the overall threshold that triggers Internet Monitor to create a health event. The default health event threshold, for both performance scores and availability scores, is 95%. That is, when the overall performance or availability score for your application falls to 95% or below, Internet Monitor creates a health event. For the overall threshold, the health event can be triggered by a single large issue, or by the combination of multiple smaller issues.

You can also change the local—that is, city-network—threshold, combined with a percentage of the overall level of impact, that—in combination—will trigger a health event. By setting a threshold that creates a health event when a score drops below the threshold for one or more city-networks (locations and ASNs, typically ISPs), you can get insights into when there are issues in locations with lower traffic, for example.

An additional local threshold option works together with the local threshold for availability or performance scores. The second factor is the percentage of your overall traffic that must be impacted before Internet Monitor creates a health event based on the local threshold.

By configuring the threshold options for overall traffic and local traffic, you can fine-tune how frequently health events are created, to align with your application usage and your needs. Be aware that when you set the local threshold to be lower, typically more health events are created, depending on your application and the other threshold configuration values that you set.

In summary, you can configure health event thresholds—for performance scores, availability scores, or both—in the following ways:

- Choose different global thresholds for triggering a health event.
- Choose different local thresholds for triggering a health event. With this option, you can also change the percentage of impact on your overall application that must be exceeded before Internet Monitor creates an event.
- Choose to turn off triggering a health event based on local thresholds, or enable local threshold options.

To update health event thresholds for performance scores, availability scores, or both, follow these steps.

To change threshold configuration options

1. In the Amazon Web Services Management Console, navigate to CloudWatch, and then, in the left navigation pane, choose Internet Monitor.
2. On the **Configure** page, in the **Health event thresholds** section, choose **Update thresholds**.
3. On the **Set health event threshold**page, choose the new values and options that you want for thresholds and other options that trigger Internet Monitor to create a health event. You can do any of the following:
   - Choose a new value for **Availability score threshold**, **Performance score threshold**, or both.

     The graphs in the sections for each setting display the current threshold setting and the actual recent health event scores, for availability or performance, for your application. By viewing the typical values, you can get an idea of values that you might want to change a threshold to.

     Tip: To view a larger graph and change the timeframe, choose the expander in the upper right corner of the graph.
   - Choose to turn on or off a local threshold for availability or performance, or both. When an option is enabled, you can set the threshold and impact level for when you want Internet Monitor to create a health event.
4. After you configure threshold options, save your updates by choosing **Update health event thresholds**.

To learn more about how health events work, see [When Internet Monitor creates and resolves health events](#).

Publish internet measurements to Amazon S3 in Internet Monitor

You can choose to have Internet Monitor publish internet measurements to Amazon S3 for your internet-facing traffic to the monitored city-networks (client locations and ASNs, typically internet service providers) in your monitor, up to the 500,000 city-networks service limit. Internet Monitor automatically publishes internet measurements to CloudWatch Logs every five minutes for the top 500 (by traffic volume) city-networks for each monitor. Measurements that it publishes to S3 include the top 500 that are published to CloudWatch Logs.

You can choose the option to publish to S3, and specify the bucket to publish the measurements, to when you create or update your monitor. The bucket must already be created in S3 before you can specify it in Internet Monitor. There's a service limit of 500,000 city-networks for internet measurements published to S3. Internet Monitor publishes internet measurements to S3 as events, a series of compressed log file objects that are stored in the bucket.

When you create the S3 bucket for Internet Monitor to publish measurements to, make sure that you follow the permissions guidance provided by CloudWatch Logs. Doing so ensures that Internet Monitor can publish logs directly to S3, and that Amazon can, if needed, create and change the resource policies associated with the log group receiving the logs. For more information, see Logs sent to CloudWatch Logs in the Amazon CloudWatch Logs User Guide.

The published log files are compressed. If you open the log files using the Amazon S3 console, they are decompressed and the internet measurement events are displayed. If you download the files, you must decompress them to view the events.

You can also query the internet measurements in the log files using Amazon Athena. Amazon Athena is an interactive query service that makes it easier to analyze data in Amazon S3, by using standard SQL. For more information, see Use Amazon Athena to query internet measurements in Amazon S3 log files.

Examples of using the CLI with Internet Monitor

This section includes examples for using the Amazon Command Line Interface with Internet Monitor operations.

Before you begin, make sure that you log in to use the Amazon CLI with the same Amazon account that has the Amazon VPC VPCs, Network Load Balancers, Amazon CloudFront distributions, or Amazon WorkSpaces directories that you want to monitor. Internet Monitor doesn't support accessing resources across accounts. For more information about using the Amazon CLI, see the

Amazon CLI Command Reference. For more information about using API actions with Internet Monitor, see the Internet Monitor API Reference Guide.

Topics

Create a monitor

When you create a monitor in Internet Monitor, you provide a name and associate resources with the monitor to show where your application's internet traffic is. You specify a traffic percentage that defines how much of your application traffic is monitored. That also determines the number of city-networks, that is, client locations and ASNs, typically internet service providers or ISPs, that are monitored. You can also opt to set a limit for the maximum number of city-networks to monitor for your application resources, to help control your bill. For more information, see Choose a city-networks maximum limit.

Finally, you can choose if you want to publish all internet measurements for your application to Amazon S3. Internet measurements for the top 500 city-networks (by traffic volume) are automatically published to CloudWatch Logs by Internet Monitor, but you can choose to publish all measurements to S3 as well.

To create a monitor with the Amazon CLI, you use the `create-monitor` command. The following command creates a monitor that monitors 100% of traffic but sets a maximum city-networks limit of 10,000, adds a VPC resource, and opts to publish internet measurements to Amazon S3.

> **Note**
>
> Internet Monitor publishes to CloudWatch Logs internet measurements every five minutes for the top 500 city-networks (client locations and ASNs, typically internet service providers or ISPs) that send traffic to each monitor. Optionally, you can choose to publish internet measurements for all monitored city-networks (up to the 500,000 city-networks service

limit) to an Amazon S3 bucket. For more information, see Publish internet measurements to Amazon S3 in Internet Monitor.

```
aws internetmonitor create-monitor --monitor-name "TestMonitor" \
    --traffic-percentage-to-monitor 100 \
    --max-city-networks-to-monitor 10000 \
    --resources "arn:aws:ec2:us-east-1:111122223333:vpc/vpc-11223344556677889" \
    --internet-measurements-log-delivery S3Config="{BucketName=amzn-s3-demo-
bucket,LogDeliveryStatus=ENABLED}"
```

```
{
    "Arn": "arn:aws:internetmonitor:us-east-1:111122223333:monitor/TestMonitor",
    "Status": "ACTIVE"
}
```

Note

You can't change the name of a monitor.

View monitor details

To view information about a monitor with the Amazon CLI, you use the `get-monitor` command.

```
aws internetmonitor get-monitor --monitor-name "TestMonitor"
```

```
{
    "ClientLocationType": "city",
    "CreatedAt": "2022-09-22T19:27:47Z",
    "ModifiedAt": "2022-09-22T19:28:30Z",
    "MonitorArn": "arn:aws:internetmonitor:us-east-1:111122223333:monitor/TestMonitor",
    "MonitorName": "TestMonitor",
    "ProcessingStatus": "OK",
    "ProcessingStatusInfo": "The monitor is actively processing data",
    "Resources": [
        "arn:aws:ec2:us-east-1:111122223333:vpc/vpc-11223344556677889"
    ],
    "MaxCityNetworksToMonitor": 10000,
    "Status": "ACTIVE"
```

```
}
```

List health events

When performance degrades for your application's internet traffic, Internet Monitor creates health events in your monitor. To see a list of current health events with the Amazon CLI, use the `list-health-events` command.

```
aws internetmonitor list-health-events --monitor-name "TestMonitor"
```


```
{
    "HealthEvents": [
        {
            "EventId": "2022-06-20T01-05-05Z/latency",
            "Status": "RESOLVED",
            "EndedAt": "2022-06-20T01:15:14Z",
            "ServiceLocations": [
                {
                    "Name": "us-east-1"
                }
            ],
            "PercentOfTotalTrafficImpacted": 1.21,
            "ClientLocations": [
                {
                    "City": "Lockport",
                    "PercentOfClientLocationImpacted": 60.370000000000005,
                    "PercentOfTotalTraffic": 2.01,
                    "Country": "United States",
                    "Longitude": -78.6913,
                    "AutonomousSystemNumber": 26101,
                    "Latitude": 43.1721,
                    "Subdivision": "New York",
                    "NetworkName": "YAHOO-BF1"
                }
            ],
            "StartedAt": "2022-06-20T01:05:05Z",
            "ImpactType": "PERFORMANCE",
            "EventArn": "arn:aws:internetmonitor:us-east-1:111122223333:monitor/
TestMonitor/health-event/2022-06-20T01-05-05Z/latency"
        },
        {
            "EventId": "2022-06-20T01-17-56Z/latency",
```

```
            "Status": "RESOLVED",
            "EndedAt": "2022-06-20T01:30:23Z",
            "ServiceLocations": [
                {
                    "Name": "us-east-1"
                }
            ],
            "PercentOfTotalTrafficImpacted": 1.29,
            "ClientLocations": [
                {
                    "City": "Toronto",
                    "PercentOfClientLocationImpacted": 75.32,
                    "PercentOfTotalTraffic": 1.05,
                    "Country": "Canada",
                    "Longitude": -79.3623,
                    "AutonomousSystemNumber": 14061,
                    "Latitude": 43.6547,
                    "Subdivision": "Ontario",
                    "CausedBy": {
                        "Status": "ACTIVE",
                        "Networks": [
                            {
                                "AutonomousSystemNumber": 16509,
                                "NetworkName": "Amazon.com"
                            }
                        ],
                        "NetworkEventType": "AWS"
                    },
                    "NetworkName": "DIGITALOCEAN-ASN"
                },
                {
                    "City": "Lockport",
                    "PercentOfClientLocationImpacted": 22.91,
                    "PercentOfTotalTraffic": 2.01,
                    "Country": "United States",
                    "Longitude": -78.6913,
                    "AutonomousSystemNumber": 26101,
                    "Latitude": 43.1721,
                    "Subdivision": "New York",
                    "NetworkName": "YAHOO-BF1"
                },
                {
                    "City": "Hangzhou",
                    "PercentOfClientLocationImpacted": 2.88,
```

```
                    "PercentOfTotalTraffic": 0.7799999999999999,
                    "Country": "China",
                    "Longitude": 120.1612,
                    "AutonomousSystemNumber": 37963,
                    "Latitude": 30.2994,
                    "Subdivision": "Zhejiang",
                    "NetworkName": "Hangzhou Alibaba Advertising Co.,Ltd."
                }
            ],
            "StartedAt": "2022-06-20T01:17:56Z",
            "ImpactType": "PERFORMANCE",
            "EventArn": "arn:aws:internetmonitor:us-east-1:111122223333:monitor/
TestMonitor/health-event/2022-06-20T01-17-56Z/latency"
        },
        {
            "EventId": "2022-06-20T01-34-20Z/latency",
            "Status": "RESOLVED",
            "EndedAt": "2022-06-20T01:35:04Z",
            "ServiceLocations": [
                {
                    "Name": "us-east-1"
                }
            ],
            "PercentOfTotalTrafficImpacted": 1.15,
            "ClientLocations": [
                {
                    "City": "Lockport",
                    "PercentOfClientLocationImpacted": 39.45,
                    "PercentOfTotalTraffic": 2.01,
                    "Country": "United States",
                    "Longitude": -78.6913,
                    "AutonomousSystemNumber": 26101,
                    "Latitude": 43.1721,
                    "Subdivision": "New York",
                    "NetworkName": "YAHOO-BF1"
                },
                {
                    "City": "Toronto",
                    "PercentOfClientLocationImpacted": 29.770000000000003,
                    "PercentOfTotalTraffic": 1.05,
                    "Country": "Canada",
                    "Longitude": -79.3623,
                    "AutonomousSystemNumber": 14061,
                    "Latitude": 43.6547,
```

```
                    "Subdivision": "Ontario",
                    "CausedBy": {
                        "Status": "ACTIVE",
                        "Networks": [
                            {
                                "AutonomousSystemNumber": 16509,
                                "NetworkName": "Amazon.com"
                            }
                        ],
                        "NetworkEventType": "AWS"
                    },
                    "NetworkName": "DIGITALOCEAN-ASN"
                },
                {
                    "City": "Hangzhou",
                    "PercentOfClientLocationImpacted": 2.88,
                    "PercentOfTotalTraffic": 0.7799999999999999,
                    "Country": "China",
                    "Longitude": 120.1612,
                    "AutonomousSystemNumber": 37963,
                    "Latitude": 30.2994,
                    "Subdivision": "Zhejiang",
                    "NetworkName": "Hangzhou Alibaba Advertising Co.,Ltd."
                }
            ],
            "StartedAt": "2022-06-20T01:34:20Z",
            "ImpactType": "PERFORMANCE",
            "EventArn": "arn:aws:internetmonitor:us-east-1:111122223333:monitor/
TestMonitor/health-event/2022-06-20T01-34-20Z/latency"
        }
    ]
}
```


View specific health event

To see a more detailed information about a specific health event with the CLI, run the `get-health-event` command with your monitor name and a health event ID.

```
aws internetmonitor get-monitor --monitor-name "TestMonitor" --event-id "health-event/
TestMonitor/2021-06-03T01:02:03Z/latency"
```

```
{
```

```
    "EventId": "2022-06-20T01-34-20Z/latency",
    "Status": "RESOLVED",
    "EndedAt": "2022-06-20T01:35:04Z",
    "ServiceLocations": [
        {
            "Name": "us-east-1"
        }
    ],
    "EventArn": "arn:aws:internetmonitor:us-east-1:111122223333:monitor/TestMonitor/
health-event/2022-06-20T01-34-20Z/latency",
    "LastUpdatedAt": "2022-06-20T01:35:04Z",
    "ClientLocations": [
        {
            "City": "Lockport",
            "PercentOfClientLocationImpacted": 39.45,
            "PercentOfTotalTraffic": 2.01,
            "Country": "United States",
            "Longitude": -78.6913,
            "AutonomousSystemNumber": 26101,
            "Latitude": 43.1721,
            "Subdivision": "New York",
            "NetworkName": "YAHOO-BF1"
        },
        {
            "City": "Toronto",
            "PercentOfClientLocationImpacted": 29.770000000000003,
            "PercentOfTotalTraffic": 1.05,
            "Country": "Canada",
            "Longitude": -79.3623,
            "AutonomousSystemNumber": 14061,
            "Latitude": 43.6547,
            "Subdivision": "Ontario",
            "CausedBy": {
                "Status": "ACTIVE",
                "Networks": [
                    {
                        "AutonomousSystemNumber": 16509,
                        "NetworkName": "Amazon.com"
                    }
                ],
                "NetworkEventType": "AWS"
            },
            "NetworkName": "DIGITALOCEAN-ASN"
        },
```

```
        {
            "City": "Shenzhen",
            "PercentOfClientLocationImpacted": 4.07,
            "PercentOfTotalTraffic": 0.61,
            "Country": "China",
            "Longitude": 114.0683,
            "AutonomousSystemNumber": 37963,
            "Latitude": 22.5455,
            "Subdivision": "Guangdong",
            "NetworkName": "Hangzhou Alibaba Advertising Co.,Ltd."
        },
        {
            "City": "Hangzhou",
            "PercentOfClientLocationImpacted": 2.88,
            "PercentOfTotalTraffic": 0.7799999999999999,
            "Country": "China",
            "Longitude": 120.1612,
            "AutonomousSystemNumber": 37963,
            "Latitude": 30.2994,
            "Subdivision": "Zhejiang",
            "NetworkName": "Hangzhou Alibaba Advertising Co.,Ltd."
        }
    ],
    "StartedAt": "2022-06-20T01:34:20Z",
    "ImpactType": "PERFORMANCE",
    "PercentOfTotalTrafficImpacted": 1.15
}
```


View monitor list

To see a list of all monitors in your account with the CLI, run the `list-monitors` command.

```
aws internetmonitor list-monitors
```


```
{
    "Monitors": [
        {
            "MonitorName": "TestMonitor",
            "ProcessingStatus": "OK",
            "Status": "ACTIVE"
        }
    ],
    "NextToken": " zase12"
```

```
}
```

Edit monitor

To update information about your monitor by using the CLI, use the `update-monitor` command and specify the name of the monitor to update. For example, you can update the percentage of traffic to monitor, the limit of the maximum number of city-networks to monitor, add or remove the resources that Internet Monitor uses to monitor traffic, and change the monitor status from `ACTIVE` to `INACTIVE`, or vice versa. Note that you can't change the name of the monitor.

The response for an `update-monitor` call returns just the `MonitorArn` and the `Status`.

The following example shows how to use the `update-monitor` command to change the maximum number of city-networks to monitor to *50000*:

```
aws internetmonitor update-monitor --monitor-name "TestMonitor" --max-city-networks-to-
monitor 50000
```

```
{
    "MonitorArn": "arn:aws:internetmonitor:us-east-1:111122223333:monitor/TestMonitor",
    "Status": " ACTIVE "
}
```

The following example shows how to add and remove resources:

```
aws internetmonitor update-monitor --monitor-name "TestMonitor" \
    --resources-to-add "arn:aws:ec2:us-east-1:111122223333:vpc/vpc-11223344556677889" \
    --resources-to-remove "arn:aws:ec2:us-east-1:111122223333:vpc/vpc-2222444455556666"
```

```
{
    "MonitorArn": "arn:aws:internetmonitor:us-east-1:111122223333:monitor/TestMonitor",
    "Status": "ACTIVE"
}
```

The following example shows how to use the `update-monitor` command to change the monitor status to `INACTIVE`:

```
aws internetmonitor update-monitor --monitor-name "TestMonitor" --status "INACTIVE"
```

```
{
```

```
    "MonitorArn": "arn:aws:internetmonitor:us-east-1:111122223333:monitor/TestMonitor",
    "Status": "INACTIVE"
}
```

Delete monitor

You can delete a monitor with the CLI by using the `delete-monitor` command. First, you must set the monitor to be inactive. To do that, use the `update-monitor` command to change the status to `INACTIVE`. Confirm that the monitor is inactive by using the `get-monitor` command and checking the status.

When the monitor status is `INACTIVE`, then you can use the CLI to run the `delete-monitor` command to delete the monitor. The response for a successful `delete-monitor` call is empty.

```
aws internetmonitor delete-monitor --monitor-name "TestMonitor"
```

```
{}
```

Monitor and optimize with the Internet Monitor dashboard

Using the Internet Monitor dashboard in the Amazon Web Services Management Console, you can visualize data and get insights and suggestions about your Amazon application's internet traffic, and configure options for your monitor.

After you create a monitor to monitor your application's internet performance and availability, Internet Monitor stores internet measurements for pairs of your client locations and ASNs, or *city-networks*. Internet Monitor also creates aggregated CloudWatch metrics for traffic to your application, and to each Amazon Web Services Region and edge location. You can filter, explore, and get action-oriented suggestions from your monitor's information in several different ways. The Internet Monitor dashboard guides you through viewing and getting insights about the data for your monitored traffic.

To get started, on the CloudWatch console, under **Network Monitoring**, choose **Internet monitors**. Then, select a monitor to work with.

> **Note**
>
> This section primarily describes how to filter and view Internet Monitor metrics using the Amazon Web Services Management Console. Alternatively, you can use Internet Monitor

API operations with the Amazon CLI or an SDK to work directly with Internet Monitor events stored in CloudWatch Logs files. For more information, see Using your monitor and measurements information. For more information about using API operations, see Examples of using the CLI with Internet Monitor and the Internet Monitor API Reference.

There are five pages (tabs) in the Internet Monitor dashboard:

- On the **Overview** page, you can get an overall view of your monitored traffic, including current performance and availability information, a summary of recent and current health events, and the top suggestion for potentially improving performance for your clients.
- On the **Health events** page, you can see current and historical health events that currently impact, or previously impacted, locations where clients access your application.
- On the **Analyze** page, you can view information about top monitored traffic in client locations (by traffic volume), summarized in several customizable ways. You can also see historical trends for health scores and metrics. You can filter by location, ASN, date, and so on, and visualize metrics for your internet traffic over time.
- On the **Optimize** page, Internet Monitor predicts your application's performance improvement for top Amazon Web Services Regions (or Amazon CloudFront), based on your traffic patterns and past performance. For each top configuration, associated tables provide a breakdown of reduced latency by client location. On a second page, you can select multiple Regions (and, if you like, include CloudFront configurations) to compare latency reductions. For each configuration (Region) that you selected, the page displays an associated table of latency details, listed by city location.
- On the **Configure** page, you can see monitor details and configure options, such as the percentage of traffic to monitor.

In addition to these dashboard options, you can use tools for deeper dives into details of the metrics that Internet Monitor collects with your monitor. Internet Monitor generates and publishes log files with the measurements about your traffic, so you can use other CloudWatch tools in the console to further visualize the data published by Internet Monitor, including CloudWatch Contributor Insights, CloudWatch Metrics, and CloudWatch Logs Insights. For more information, see Exploring your data with CloudWatch tools and the Internet Monitor query interface.

Learn about using Internet Monitor to explore your performance and availability measurements in the following sections.

Topics

Track real-time performance and availability in Internet Monitor (Overview page)

The **Overview** page in the Internet Monitor console shows you a high-level view of performance and availability for the traffic that your monitor tracks, and a timeline of when health events have impacted your monitored traffic. The page also provides the top suggestion for a configuration change that could reduce latency for clients who use your application in your top client location (by traffic volume).

Traffic overview and Status

The **Traffic overview** section provides an overall look at your application's availability and performance. Note that this section shows *aggregated* overall performance and availability scores that consider all of the traffic for applications towards all end users and service locations. You can see health scores for specific client locations and service locations by searching and filtering measurements information on the **Analyze** tab.

Under **Status**, you can see if your monitor is actively creating data for your monitor or is waiting for data to be available. You can also see the percentage of your application's traffic that you're monitoring. If you want to change the percentage, see the **Configure** page.

Internet Monitor uses a statistical process to create availability and performance scores for your monitored traffic. Amazon has substantial historical data about internet performance and availability for network traffic between geographic locations for different ASNs and Amazon services. Internet Monitor uses the connectivity data that Amazon has captured from its global networking footprint to calculate a baseline of availability and performance for internet traffic. This is the same data that we use at Amazon to monitor our own internet uptime and availability.

With those measurements as a baseline, Internet Monitor can detect when the performance and availability for your application has dropped, compared to the baseline. To make it easier to see those drops, we report that information to you as a performance score and an availability score.

For more information, see How Amazon calculates performance and availability scores.

Health events timeline

The **Health events timeline** graph displays health events that have occurred during the past 24 hours. A summary below the graph shows your application's current and recent impact. For details, you can choose **See more health events**.

To change the thresholds for health events, go to the **Configure** page.

Reduce latency for your top Region

Internet Monitor automatically evaluates the Amazon Web Services Region that your current application configuration uses most (that is, the Region with the highest client volume), and determines if another Region could provide a better aggregate time to first byte (TTFB) for your clients.

Note that because this is the aggregate TTFB, if you move traffic from one Region to another, TTFB for most locations is expected to improve but clients in some Regions could see no change or reduced performance..

To explore more latency improvement suggestions, including details at more granular levels (such as by client location), see the **Optimize** page.

View health events and metrics in Internet Monitor (Health events page)

The **Health events** page in the Internet Monitor console provides a map of health events that impact the client locations and ASNs for your application. You can click circles on the map for more details about an event. The **Health events** tables lists locations that have been impacted by an event, and specifics about the impact.

Internet traffic overview

The **Internet traffic overview** map shows you the internet traffic and health events that are specific to the locations and ASNs that your clients access your application from. The countries that are gray on the map are those that include traffic for your application.

Each circle on the map indicates a health event in an area, for a time period that you select. Internet Monitor creates health events when it detects a problem, at a specific (but customizable) threshold, with connectivity between one of your resources hosted in Amazon and a city-network where a client is accessing your application.

Choose a circle on the map to display more details about the health event for that location. In addition, for clusters that have health events, you can see detailed information in the **Health events** table below the map.

Note that Internet Monitor creates health events in a monitor when it determines that an event has significant impact on your application. The map is blank if there aren't any health events that exceed the threshold for impact on traffic for your client locations in the time period that you've selected. For more information, see When Internet Monitor creates and resolves health events.

Health events

The **Health events** table lists client locations that have been affected by health events, along with information about the events. The following columns are included in the table.

Column	Description
Event type	Specifies whether current events are *overall* health events or *local* health events, or if the health event is in the *past*.
Client location	The location of the end users who were impacted by the event, who experienced increased latency or reduced availability. To learn more about client location accuracy in Internet Monitor, see Geolocation information and accuracy in Internet Monitor.
ISP name (ASN)	The network that the traffic traveled over. Typically, this is the internet service provider (ISP) or Autonomous System Number (ASN) for the network traffic.
Service location	The Amazon location for the network traffic, which can be an Amazon Web Services Region or an internet edge location.

Column	Description
Traffic impact	How much impact was caused by the event, in increased latency or reduced availability. For latency, this is the percentage of how much latency increased during the event compared to typical performance for traffic, from this client location to this Amazon location using this client network.
Impact type	The type of impact for the health event. Health events are typically caused by latency increases (performance issues) or reachabil ity (availability issues). You might also be able to click on the impact type to see the cause of the impairment. When possible, Internet Monitor analyzes the origin of a health event, to determine whether it was caused by Amazon or an ASN (internet service provider). Note that this analysis continues after the event is resolved. Internet Monitor can update events with new information for up to an hour.

If you choose one of the client locations in the **Health events** table, you can see more details about the health event at that location. For example, you can see when the event started, when it ended, and the local traffic impact.

Network path visualization

If Internet Monitor has finished impairment analysis for an event, you can view **Network path visualization** to see the full network path for traffic to a client location. The full path shows you each node along the network path for your application for the health event, between the Amazon location and the client, for a client-location pair.

When Internet Monitor has determined the cause of an impairment, Internet Monitor adds a dashed red circle around the node. Impairments can be caused by ASNs, typically internet service providers (ISPs), or the cause can be Amazon. If there were multiple causes for an impairment, multiple nodes are circled.

Analyze historical data in Internet Monitor (Analyze page)

On the **Analyze** page in the Internet Monitor console, you can view your application's the top client locations for the traffic that you monitor, by traffic volume. You can also view graphs showing performance and availability scores for your traffic over time, as well as graphs of other internet traffic metrics for your application's monitored traffic.

To start exploring Internet Monitor data for your application traffic, select a time period. Then, choose a specific geographical location, such a city, and (optionally) other filters. Internet Monitor applies the filters to your data, and then you can see graphs of the data that show measurements for your application. The graphs included on the **Analyze** page include your application's performance score, availability score, monitored bytes transferred (for VPCs, Network Load Balancers, and CloudFront distributions) or client connection counts (for WorkSpaces directories), and round-trip time (RTT) for your application over time.

The options at the top of the **Analyze** page determine the timeframe and types of traffic shown in the graphs on the page. You can filter by client locations or ASN, or choose to show traffic graphs at a specific granularity (the default is city level).

Top client locations

The **Top client locations** graph displays your top monitored traffic locations, by default. You can choose another field to sort the graph by, or you can sort the graph in other ways, for example, by lowest traffic locations.

The filters that you choose for the page determine the Regions, timeframe, and so on for the locations.

Traffic health scores

This section shows you graphs of traffic health scores and metrics for your monitored traffic. These graphs reflect data for the filters that you choose at the top of the page.

The **Traffic health scores** graph shows you performance and availability information for your local and overall traffic by calling out health events that have impacted your monitored client traffic. Amazon has substantial historical data about internet performance and availability for network traffic between geographic locations for different ASNs and Amazon services. Internet Monitor uses the connectivity data that Amazon has captured from its global networking footprint to calculate a baseline of performance and availability for internet traffic. This is the same data that we use at Amazon to monitor our own internet uptime and availability.

With those measurements as a baseline, Internet Monitor can detect when the performance and availability for your application has dropped, compared to the baseline. To make it easier to see those drops, we report that information to you as a performance score and an availability score. For more information, see How Amazon calculates performance and availability scores.

Additional graphs show the monitored bytes transferred (for VPCs, Network Load Balancers, and CloudFront distributions) or client connection counts (for WorkSpaces directories), and round-trip time (RTT) for your application traffic.

Note that when round-trip time (RTT) is aggregated across end-user locations, the value is weighted by the amount of your traffic that is driven by each client location. For example, with two client locations, one serving 90% of traffic with a 5 ms RTT, and the other serving 10% of traffic with a 10 ms RTT, the result is an aggregated RTT of 5.5 ms (which comes from 5 ms * 0.9 + 10 ms * 0.1).

You can also explore the internet measurements that Internet Monitor stores for your monitored traffic by using CloudWatch tools or other methods. For more information, see Exploring your data with CloudWatch tools and the Internet Monitor query interface. In addition, you can create CloudWatch alarms based on Internet Monitor data, for example, to notify you of health events. For more information, see Create alarms with Internet Monitor.

Get suggestions to optimize application performance in Internet Monitor (Optimize page)

Use the **Optimize** page in the Internet Monitor console to get suggestions for how to optimize application performance for your clients. Internet Monitor evaluates your monitored application traffic, and determines if you can reduce latency by changing the Amazon Web Services Regions that you've configured for your application. Optionally, you can also view latency changes if you choose to include Amazon CloudFront in the suggestions.

You can review suggestions for your application's top Regions by traffic volume, or for top client locations, also by traffic volume.

Suggestions to reduce latency for top Regions

To help you quickly understand your best options for reducing latency for your clients, Internet Monitor automatically provides suggestions for improving latency in your application for your top Regions (by traffic volume).

You can also explore configuration changes for all the Regions where your application serves clients. This includes getting details about each suggested change at deeper levels of granularity, for example, by specific client location. To explore all Regional configurations and expected latency changes for your application, choose **Optimization suggestions for all Regions**.

Suggestions to reduce latency for all Regions

To explore suggestions for reducing latency for all Regions where clients access your application, choose **Optimization suggestions for all Regions** to open a new dashboard page. On this page, you can select different Regions to configure, with the option of including CloudFront as a configuration comparison, and then compare the times to first byte (TTFBs) for each selected configuration.

Then, for each comparison, you can also see a table with at a more granular level (by client location), with the average expected TTFB for each one.

Suggestions to reduce latency for top locations

Internet Monitor also provides suggestions for reducing application latency for your clients by specific location. When the table lists multiple suggestions for the same location, expand the city location for that row to see details.

Be aware that if you change a configuration to use a different Region or to use CloudFront, latency improvements can vary by client location. For example, latency might improve for some locations, but stay the same or worsen for others.

Suggestions to reduce latency by updating routing configurations

Note: These suggestions are only relevant for application traffic to Regional load balancers. The table is not shown for monitors that you create for CloudFront distributions or WorkSpaces resources.

With Internet Monitor, you can view information about latency toward Amazon locations for IPv4 IP prefixes that access your application using different DNS resolvers (typically ISPs). Using this information, you can take steps to reduce latency for specific groups of users by routing a set of IP address prefixes, specified by a CIDR collection, to your endpoints in a Region that results in lower latency for your users. If you don't already have a CIDR collection for the prefixes, you can go to Amazon Route 53 to create one. Then, you can update your routing in Route 53 to route IP addresses in the collection to a specific Region.

If you want to create a CIDR collection for a set of IP address prefixes, you can easily do so by selecting a row or rows that includes the IP prefixes that you want, and then choosing **Add to CIDR collection**. Then, in the Route 53 console, you can configure a routing policy that routes IP addresses in the collection to the Region with lower latency for your application.

To learn more about IP-based routing in Route 53, see IP-based routing.

By viewing the suggestions on this page, you can start planning configurations and deployments that can improve performance for your clients. Note that you might see a dash (-) instead of a value in a column, when data is not available to display.

For more information about TTFB calculations, see Amazon calculations for TTFB and latency. To review a specific example of how to improve performance, see Using Internet Monitor for a Better Gaming Experience.

Monitoring details in Internet Monitor (Configure page)

On the **Configure** page, you can see details about your monitor, including a list of resources that you monitor traffic for and the thresholds for when health events are triggered. You can also explore and compare values for the traffic percentage for your monitor, and its impact on how many city-networks are included for (monitored by) your monitor. Finally, you can view information about measurements that are published to an Amazon S3 bucket.

You can configure a monitor to change most options, such as the percentage of traffic to monitor. For more information, see Configure your monitor.

Monitor details

The **Monitor details** section includes basic information about your monitor, including the name, the percentage of traffic currently being monitored for your application, a city-networks maximum limit (if you've set one), and status information about the monitor.

The following explains the values that you might see for **Status** and **Status info** (data processing status).

Status	Description
Active	Monitor is created and active.
Pending	Monitor is currently being created and is not yet active.
Inactive	Monitor is created but has been set to inactive.
Error	Monitor is in an error state.

Status details (Data processing status)	Description
OK	Monitor is actively processing data.
Inactive	Monitor is inactive and is not processing data.
Collecting data	Monitor is actively collecting data.
Insufficient data	Monitor is actively processing data, but there aren't enough datapoints to produce insights.
Fault access CloudWatch	Monitor has encountered a problem delivering CloudWatch metrics data and log events.

Health event thresholds

In this section, you can see the current thresholds for health events that are configured for this monitor. If you haven't configured any custom thresholds, the values shown here are the default values.

By default, health events are not triggered based on local thresholds. If local health event thresholds would be useful for your Internet Monitor scenario, you can enable the option and specify the thresholds to use.

You can learn more about how health event thresholds work, and review the potential impact of adding local thresholds or changing existing thresholds. For more information, see Change health event thresholds.

Traffic coverage

In this section, you can explore options for the traffic coverage for your monitor. When you change the traffic percentage for a monitor, Internet Monitor monitors different amounts of application traffic. If you set a traffic percentage to less than 100% (100% is the default value), some portion of the city-networks that your clients use to access your application might not be monitored. By exploring the impact of different traffic percentage values, you can see how different values that you might set would impact your city-networks coverage.

The **Total monitored city-networks** graph shows you how many city-networks are currently monitored, and how many would be monitored if you set the traffic percentage to 100%. To view different traffic percentage values on the graph, select percentages in the drop-down menu.

After you explore the options, you can change the traffic percentage to monitor by choosing **Update monitoring coverage**.

If you want to set a maximum city-networks limit, at the top of the page, choose **Edit monitor**. Then, under **Advanced options**, set a maximum city-networks value.

Configure your monitor

As on every page in the Internet Monitor dashboard, you can choose **Edit monitor** to change options for your monitor, including adding or removing resources. For details about how to update the following configuration options, see the provided links.

View health event thresholds

> In this section, you can see the current thresholds for health events that are configured for this monitor.
>
> To update health thresholds, see Change health event thresholds.

View and evaluate traffic coverage

In this section, you can compare the effect of changing the percentage of traffic that you monitor for your application on the number of city-networks that are included (for monitoring) when you choose different percentage values.

You can also change the percentage of traffic that you monitor, or change the limit for the number of city-networks your monitor includes. To change the percentage of traffic, choose **Update monitoring coverage**.

For detailed steps and information, see Explore changing your application traffic percentage.

Configuration details for publishing internet measurements to Amazon S3

If you have configured Internet Monitor to publish internet measurements for your monitor to an Amazon S3 bucket, the information about your configuration is shown here.

To configure this option, see Publishing internet measurements to S3.

Exploring your data with CloudWatch tools and the Internet Monitor query interface

In addition to visualizing your performance and availability for your application with the Internet Monitor dashboard, there are several methods that you can use to dive deeper into the data that Internet Monitor generates for you. These methods include using CloudWatch tools with Internet Monitor data stored in CloudWatch Log files and using the Internet Monitor query interface. The tools that you can use include CloudWatch Logs Insights, CloudWatch Metrics, CloudWatch Contributor Insights, and Amazon Athena. You can use some or all these tools, as well as the dashboard, to explore Internet Monitor data, depending on your needs.

Internet Monitor aggregates CloudWatch metrics about traffic to your application and to each Amazon Web Services Region, and includes data such as total traffic impact, availability, and round-trip time. This data is published to CloudWatch Logs and is also available to use with the Internet Monitor query interface. Details about geo-granularity and other aspects of the information available to explore for each one varies.

Internet Monitor publishes data for your monitor at 5 minute intervals, and then makes the data available in several ways. The following table lists scenarios for accessing Internet Monitor data, and describes features of the data that is collected for each one.

Feature	CloudWatch Logs	Export to S3	Query interface	CloudWatch dashboard
Enabled by default	Yes	No	Yes	Yes
Number of city-networks that data is collected for	Top 500 (see note below)	All	All	All
Data retention	User controlled	User controlled	30 days	30 days
Geo-granularities that data is collected for	All (city-network, metro+network, subdivision+network, country+network)	City-network	All (city-network, metro+network, subdivision+network, country+network)	All (city-network, metro+network, subdivision+network, country+network)
How to query and filter data	Use CloudWatch Logs Insights to explore Internet Monitor measurements	Use Amazon Athena to query internet measurements in Amazon S3 log files	Use the Internet Monitor query interface	Monitor and optimize with the Internet Monitor dashboard

Note: Top 500 measurements are captured for city-networks; top 250 for metro+networks, top 100 for subdivision+networks, top 50 for country+networks.

This chapter describes how to query and explore your data by using CloudWatch tools or the Internet Monitor query interface, together with examples for each method.

Contents

- Use Amazon Athena to query internet measurements in Amazon S3 log files
- Use the Internet Monitor query interface

Use CloudWatch Logs Insights to explore Internet Monitor measurements

You can use CloudWatch Logs Insights queries to filter a subset of logs for a specific city or geography (client location), client ASN (ISP), and Amazon source location. Internet Monitor publishes granular measurements of availability and round-trip time to CloudWatch Logs that you can explore using CloudWatch Logs Insights.

To learn more about client location accuracy in Internet Monitor, see Geolocation information and accuracy in Internet Monitor.

The examples in this section can help you create CloudWatch Logs Insights queries to learn more about your own application traffic measurements and metrics. If you use these examples in CloudWatch Logs Insights, replace *monitorName* with your own monitor name.

View traffic optimization suggestions

On the **Traffic insights** tab in Internet Monitor, you can view traffic optimization suggestions, filtered by a location. To see the same information that is displayed in the **Traffic optimization suggestions** section on that tab, but without the location granularity filter, you can use the following CloudWatch Logs Insights query.

1. In the Amazon Web Services Management Console, navigate to CloudWatch Logs Insights.
2. For **Log Group**, select `/aws/internet-monitor/monitorName/byCity` and `/aws/internet-monitor/monitorName/byCountry`, and then specify a time range.
3. Add the following query, and then run the query.

```
fields @timestamp,
clientLocation.city as @city, clientLocation.subdivision as @subdivision,
 clientLocation.country as @country,
`trafficInsights.timeToFirstByte.currentExperience.serviceName` as @serviceNameField,
concat(@serviceNameField, ` (`, `serviceLocation`, `)`) as @currentExperienceField,
concat(`trafficInsights.timeToFirstByte.ec2.serviceName`, ` (`,
 `trafficInsights.timeToFirstByte.ec2.serviceLocation`, `)`) as @ec2Field,
`trafficInsights.timeToFirstByte.cloudfront.serviceName` as @cloudfrontField,
concat(`clientLocation.networkName`, ` (AS`, `clientLocation.asn`, `)`) as @networkName
```

```
| filter ispresent(`trafficInsights.timeToFirstByte.currentExperience.value`)
| stats avg(`trafficInsights.timeToFirstByte.currentExperience.value`) as @averageTTFB,
avg(`trafficInsights.timeToFirstByte.ec2.value`) as @ec2TTFB,
avg(`trafficInsights.timeToFirstByte.cloudfront.value`) as @cloudfrontTTFB,
sum(`bytesIn` + `bytesOut`) as @totalBytes,
latest(@ec2Field) as @ec2,
latest(@currentExperienceField) as @currentExperience,
latest(@cloudfrontField) as @cloudfront,
count(*) by @networkName, @city, @subdivision, @country
| display @city, @subdivision, @country, @networkName, @totalBytes, @currentExperience,
 @averageTTFB, @ec2, @ec2TTFB, @cloudfront, @cloudfrontTTFB
| sort @totalBytes desc
```

View internet availability and RTT (p50, p90, and p95)

To view the internet availability and round-trip time (p50, p90, and p95) for traffic, you can use the following CloudWatch Logs Insights query.

End user geography: Chicago, IL, United States

End user network (ASN): AS7018

Amazon service location: US East (N. Virginia) Region

To view the logs, do the following:

1. In the Amazon Web Services Management Console, navigate to CloudWatch Logs Insights.
2. For **Log Group**, select `/aws/internet-monitor/`*`monitorName`*`/byCity` and `/aws/internet-monitor/`*`monitorName`*`/byCountry`, and then specify a time range.
3. Add the following query, and then run the query.

The query returns all the performance data for users connecting from AS7018 in Chicago, IL towards US East (N. Virginia) Region over the selected time period.

```
fields @timestamp,
internetHealth.availability.experienceScore as availabilityExperienceScore,
internetHealth.availability.percentageOfTotalTrafficImpacted as
 percentageOfTotalTrafficImpacted,
internetHealth.performance.experienceScore as performanceExperienceScore,
internetHealth.performance.roundTripTime.p50 as roundTripTimep50,
internetHealth.performance.roundTripTime.p90 as roundTripTimep90,
```

```
internetHealth.performance.roundTripTime.p95 as roundTripTimep95
 | filter clientLocation.country == `United States`
 and clientLocation.city == `Chicago`
 and serviceLocation == `us-east-1`
 and clientLocation.asn == 7018
```

For more information, see Analyzing log data with CloudWatch Logs Insights.

Use Contributor Insights to identify top locations and ISPs

CloudWatch Contributor Insights can help you identify top client locations and ASNs (typically, internet service providers or ISPs) for your Amazon application. Use the following sample Contributor Insights rules to get started with rules that are useful with Internet Monitor. For more information, see Create a Contributor Insights rule in CloudWatch.

To learn more about client location accuracy in Internet Monitor, see Geolocation information and accuracy in Internet Monitor.

> **Note**
>
> Internet Monitor stores internet measurements data every five minutes, so after you set up a Contributor Insights rule, you must adjust the period to five minutes to see a graph.

View top locations and ASNs impacted by an availability impact

To view top client locations and ASNs impacted by a drop in availability, you can use the following Contributor Insights rule in the Syntax editor. Replace *monitor-name* with your own monitor name.

```
{
    "Schema": {
        "Name": "CloudWatchLogRule",
        "Version": 1
    },
    "AggregateOn": "Sum",
    "Contribution": {
        "Filters": [
            {
                "Match": "$.clientLocation.city",
                "IsPresent": true
```

```
            }
        ],
        "Keys": [
            "$.clientLocation.city",
            "$.clientLocation.networkName"
        ],
        "ValueOf": "$.awsInternetHealth.availability.percentageOfTotalTrafficImpacted"
    },
    "LogFormat": "JSON",
    "LogGroupNames": [
        "/aws/internet-monitor/monitor-name/byCity"
    ]
}
```

View top client locations and ASNs impacted by a latency impact

To view top client locations and ASNs impacted by an increase in round-trip time (latency), you can use the following Contributor Insights rule in the Syntax editor. Replace *`monitor-name`* with your own monitor name.

```
{
    "Schema": {
        "Name": "CloudWatchLogRule",
        "Version": 1
    },
    "AggregateOn": "Sum",
    "Contribution": {
        "Filters": [            {
                "Match": "$.clientLocation.city",
                "IsPresent": true
            }
        ],
        "Keys": [
            "$.clientLocation.city",
            "$.clientLocation.networkName"
        ],
        "ValueOf": "$.awsInternetHealth.performance.percentageOfTotalTrafficImpacted"
    },
    "LogFormat": "JSON",
    "LogGroupNames": [
        "/aws/internet-monitor/monitor-name/byCity"
    ]
}
```

View top client locations and ASNs impacted by total percentage of traffic

To view top client locations and ASNs impacted by total percentage of traffic, you can use the following Contributor Insights rule in the Syntax editor. Replace *monitor-name* with your own monitor name.

```
{
    "Schema": {
        "Name": "CloudWatchLogRule",
        "Version": 1
    },
    "AggregateOn": "Sum",
    "Contribution": {
        "Filters": [
            {
                "Match": "$.clientLocation.city",
                "IsPresent": true
            }
        ],
        "Keys": [
            "$.clientLocation.city",
            "$.clientLocation.networkName"
        ],
        "ValueOf": "$.percentageOfTotalTraffic"
    },
    "LogFormat": "JSON",
    "LogGroupNames": [
        "/aws/internet-monitor/monitor-name/byCity"
    ]
}
```

View Internet Monitor metrics or set alarms in CloudWatch Metrics

You can view or set alarms on Internet Monitor metrics by using CloudWatch alarms and CloudWatch Metrics in the CloudWatch console. Internet Monitor publishes metrics to your account, including metrics for performance, availability, round-trip time, and throughput (bytes per second). To find all metrics for your monitor, in the CloudWatch Metrics dashboard, see the custom namespace `AWS/InternetMonitor`.

To see examples for using several of these metrics to help determine values to choose for a city-networks maximum limit for your monitor, see Choosing a city-network maximum value. To learn more about setting alarms for Internet Monitor, see Create alarms with Internet Monitor.

Metrics are aggregated across all internet traffic to your VPCs, Network Load Balancers, CloudFront distributions, or WorkSpaces directories in the monitor, and to all traffic to each Amazon Web Services Region and internet edge location that is monitored. Regions are defined by the service location, which can either be all locations or a specific Region, such as `us-east-1`.

Note: *city-networks* are pairs of client locations and the ASNs the clients use (typically internet service providers or ISPs).

Internet Monitor provides the following metrics.

Metric	Description
PerformanceScore	A performance score represents the estimated percentage of traffic that is not seeing a performance drop.
AvailabilityScore	An availability score represents the estimated percentage of traffic that is not seeing an availability drop.
BytesIn	Bytes transferred in for your application internet traffic at all application city-networks.
BytesOut	Bytes transferred out for your application internet traffic at all application city-networks.
BytesInMonitored	Bytes transferred in for your application internet traffic at monitored city-networks.
BytesOutMonitored	Bytes transferred out for your application internet traffic at monitored city-networks.
Round-trip time (RTT)	Round-trip time between the Amazon Web Services Regions, ASNs (typically internet service providers or ISPs), and locations (such as cities) specific to your VPCs, Network Load Balancers, CloudFront distributions, or WorkSpaces directories.

Metric	Description
CityNetworksMonitored	The number of city-networks Internet Monitor monitored for your application internet traffic. This is never more than the upper limit that you set as the maximum city-networks for the monitor.
TrafficMonitoredPercent	The percentage of total application internet traffic for this monitor that is represented (included) by the city-networks that Internet Monitor is monitoring. This is less than 100 (that is, less than 100%) if clients access your application in more city-networks than the maximum city-networks limit that you have set for the monitor.
CityNetworksFor100PercentTraffic	The number that you should set your city-networks maximum limit to if you want to monitor 100% of your application internet traffic in Internet Monitor.
CityNetworksFor99PercentTraffic	The number that you should set your city-networks maximum limit to if you want to monitor 99% of your application internet traffic in Internet Monitor.
CityNetworksFor95PercentTraffic	The number that you should set your city-networks maximum limit to if you want to monitor 95% of your application internet traffic in Internet Monitor.
CityNetworksFor90PercentTraffic	The number that you should set your city-networks maximum limit to if you want to monitor 90% of your application internet traffic in Internet Monitor.

Metric	Description
CityNetworksFor75PercentTraffic	The number that you should set your city-networks maximum limit to if you want to monitor 75% of your application internet traffic in Internet Monitor.
CityNetworksFor50PercentTraffic	The number that you should set your city-networks maximum limit to if you want to monitor 50% of your application internet traffic in Internet Monitor.
CityNetworksFor25PercentTraffic	The number that you should set your city-networks maximum limit to if you want to monitor 25% of your application internet traffic in Internet Monitor.

For more information, see Metrics in Amazon CloudWatch.

Use Amazon Athena to query internet measurements in Amazon S3 log files

You can use Amazon Athena to query and view the internet measurements that Internet Monitor publishes to an Amazon S3 bucket. There's an option in Internet Monitor to publish internet measurements for your application to an S3 bucket for internet-facing traffic for your monitored city-networks (client locations and ASNs, typically internet service providers or ISPs). Regardless of whether you choose to publish measurements to S3, Internet Monitor automatically publishes internet measurements to CloudWatch Logs every five minutes for the top 500 (by traffic volume) city-networks for each monitor.

This chapter includes steps for how to create a table in Athena for internet measurements located in an S3 log file, and then provides example queries to see different views of the measurements. For example, you can query for your top 10 impacted city-networks by latency impact.

Using Amazon Athena to create a table for internet measurements in Internet Monitor

To start using Athena with your Internet Monitor S3 log files, you first create a table for the internet measurements.

Follow the steps in this procedure to create a table in Athena based on the S3 log files. Then, you can run Athena queries on the table, such as these example internet measurements queries, to get information about your measurements.

To create an Athena table

1. Open the Athena console at https://console.aws.amazon.com/athena/.
2. In the Athena query editor, enter a query statement to generate a table with Internet Monitor internet measurements. Replace the value for the LOCATION parameter with the location of S3 bucket where your Internet Monitor internet measurements are stored.

```
CREATE EXTERNAL TABLE internet_measurements (
    version INT,
    timestamp INT,
    clientlocation STRING,
    servicelocation STRING,
    percentageoftotaltraffic DOUBLE,
    bytesin INT,
    bytesout INT,
    clientconnectioncount INT,
    internethealth STRING,
    trafficinsights STRING
)
PARTITIONED BY (year STRING, month STRING, day STRING)
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
LOCATION
's3://amzn-s3-demo-bucket/bucket_prefix/AWSLogs/account_id/
internetmonitor/AWS_Region/'
TBLPROPERTIES ('skip.header.line.count' = '1');
```

3. Enter a statement to create a partition to read the data. For example, the following query creates a single partition for a specified date and location:

```
ALTER TABLE internet_measurements
ADD PARTITION (year = 'YYYY', month = 'MM', day = 'dd')
LOCATION
's3://amzn-s3-demo-bucket/bucket_prefix/AWSLogs/account_id/
internetmonitor/AWS_Region/YYYY/MM/DD';
```

4. Choose **Run**.

Example Athena statements for internet measurements

The following is an example of a statement to generate a table:

```
CREATE EXTERNAL TABLE internet_measurements (
    version INT,
    timestamp INT,
    clientlocation STRING,
    servicelocation STRING,
    percentageoftotaltraffic DOUBLE,
    bytesin INT,
    bytesout INT,
    clientconnectioncount INT,
    internethealth STRING,
    trafficinsights STRING
)
PARTITIONED BY (year STRING, month STRING, day STRING)
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
LOCATION 's3://internet-measurements/TestMonitor/AWSLogs/1111222233332/internetmonitor/
us-east-2/'
TBLPROPERTIES ('skip.header.line.count' = '1');
```

The following is an example of a statement to create a partition to read the data:

```
ALTER TABLE internet_measurements
ADD PARTITION (year = '2023', month = '04', day = '07')
LOCATION 's3://internet-measurements/TestMonitor/AWSLogs/1111222233332/internetmonitor/
us-east-2/2023/04/07/'
```

Sample Amazon Athena queries to use with internet measurements in Internet Monitor

This section includes example queries that you can use with Amazon Athena to get information about your application's internet measurements published to Amazon S3.

Query your top 10 impacted (by total percentage of traffic) client locations and ASNs

Run this Athena query to return your top 10 impacted (by total percentage of traffic) city-networks—that is, client locations and ASNs, typically internet service providers.

```
SELECT json_extract_scalar(clientLocation, '$.city') as city,
    json_extract_scalar(clientLocation, '$.networkname') as networkName,
```

```
    sum(percentageoftotaltraffic) as percentageoftotaltraffic
FROM internet_measurements
GROUP BY json_extract_scalar(clientLocation, '$.city'),
    json_extract_scalar(clientLocation, '$.networkname')
ORDER BY percentageoftotaltraffic desc
limit 10
```

Query your top 10 impacted (by availability) client locations and ASNs

Run this Athena query to return your top 10 impacted (by total percentage of traffic) city-networks—that is, client locations and ASNs, typically internet service providers.

```
SELECT json_extract_scalar(clientLocation, '$.city') as city,
    json_extract_scalar(clientLocation, '$.networkname') as networkName,
    sum(
        cast(
            json_extract_scalar(
                internetHealth,
                '$.availability.percentageoftotaltrafficimpacted'
            )
        as double )
    ) as percentageOfTotalTrafficImpacted
FROM internet_measurements
GROUP BY json_extract_scalar(clientLocation, '$.city'),
    json_extract_scalar(clientLocation, '$.networkname')
ORDER BY percentageOfTotalTrafficImpacted desc
limit 10
```

Query your top 10 impacted (by latency) client locations and ASNs

Run this Athena query to return your top 10 impacted (by latency impact) city-networks—that is, client locations and ASNs, typically internet service providers.

```
SELECT json_extract_scalar(clientLocation, '$.city') as city,
    json_extract_scalar(clientLocation, '$.networkname') as networkName,
    sum(
        cast(
            json_extract_scalar(
                internetHealth,
                '$.performance.percentageoftotaltrafficimpacted'
            )
        as double )
```

```
    ) as percentageOfTotalTrafficImpacted
FROM internet_measurements
GROUP BY json_extract_scalar(clientLocation, '$.city'),
    json_extract_scalar(clientLocation, '$.networkname')
ORDER BY percentageOfTotalTrafficImpacted desc
limit 10
```

Query traffic highlights for your client locations and ASNs

Run this Athena query to return traffic highlights, including availability score, performance score, and time to first byte for your city-networks—that is, client locations and ASNs, typically internet service providers. .

```

SELECT json_extract_scalar(clientLocation, '$.city') as city,
    json_extract_scalar(clientLocation, '$.subdivision') as subdivision,
    json_extract_scalar(clientLocation, '$.country') as country,
    avg(cast(json_extract_scalar(internetHealth, '$.availability.experiencescore') as
 double)) as availabilityScore,
    avg(cast(json_extract_scalar(internetHealth, '$.performance.experiencescore') as
 double)) performanceScore,
    avg(cast(json_extract_scalar(trafficinsights,
 '$.timetofirstbyte.currentexperience.value') as double)) as averageTTFB,
    sum(bytesIn) as bytesIn,
    sum(bytesOut) as bytesOut,
    sum(bytesIn + bytesOut) as totalBytes
FROM internet_measurements
where json_extract_scalar(clientLocation, '$.city') != 'N/A'
GROUP BY
json_extract_scalar(clientLocation, '$.city'),
    json_extract_scalar(clientLocation, '$.subdivision'),
    json_extract_scalar(clientLocation, '$.country')
ORDER BY totalBytes desc
limit 100
```

For more information about using Athena, see the [Amazon Athena User Guide](#).

Use the Internet Monitor query interface

An option for understanding more about internet traffic for your Amazon application is to use the Internet Monitor *query interface*. To use the query interface, you create a query with data filters that you choose, and then run the query to return a subset of your Internet Monitor data. Exploring the data that the query returns can give you insights into how your application is performing on the internet.

You can query and explore all the metrics that Internet Monitor captures with your monitor, including availability and performance scores, bytes transferred, round-trip times, and time to first byte (TTFB).

Internet Monitor uses the query interface to provide the data that you can explore in the Internet Monitor console dashboard. By using search options in the dashboard—on the **Analyze** page or the **Optimize** page—you can query and filter internet data for your application.

If you'd like more flexibility to explore and filter your data than the dashboard provides, you can use the query interface yourself, by using Internet Monitor API operations with the Amazon Command Line Interface or with an Amazon SDK. This section introduces the types of queries that you can use with the query interface, and the filters that you can specify to create a subset of data, to get insights about internet traffic for your application.

Topics

How to use the query interface

You create a query with the query interface by choosing a *query type*, and then specifying filter values, to return a specific desired subset of your log file data. Then, you can work with the data subset, to further filter and sort, create reports, and so on.

The query process works like this:

1. When you run a query, Internet Monitor returns a `query ID` that is unique to the query. This section describes the query types that are available, and options for filtering data in queries. To understand how this works, you can also review the section on query examples.
2. You specify the query ID with your monitor name with the GetQueryResults API operation to return data results for the query. Each query type returns a different set of data fields. To learn more, see Get query results.

The query interface provides the following query types. Each query type returns a different set of information about your traffic from the log files, as shown.

- **Measurements:** Provides availability score, performance score, total traffic, and round-trip times, at 5 minute intervals.
- **Top locations:** Provides availability score, performance score, total traffic, and time to first byte (TTFB) information, for the top location and ASN combinations that you're monitoring, by traffic volume.
- **Top locations details:** Provides TTFB for Amazon CloudFront, your current configuration, and the best performing Amazon EC2 configuration, at 1 hour intervals.
- **Overall traffic suggestions:** Provides TTFB, using a 30-day weighted average, for all traffic in each Amazon location that is monitored.
- **Overall traffic suggestions details:** Provides TTFB, using a 30-day weighted average, for each top location, for a proposed Amazon location.
- **Routing suggestions:** Provides the predicted average round-trip time (RTT) from an IP prefix toward an Amazon location for a DNS resolver. The RTT is calculated at one hour intervals, over a one hour period.

You can filter the data more by using specific criteria. With most query types, except routing suggestions, you can filter by specifying one or more of the following criteria:

- **Amazon location:** For Amazon location, you can specify CloudFront or an Amazon Web Services Region, such as `us-east-2`.
- **ASN:** Specify the autonomous system number (ASN) of a DNS resolver (typically, an ISP), for example, 4225.
- **Client location:** For location, specify a city, metro, subdivision, or country.
- **Proposed Amazon location:** Specify an Amazon Web Services Region, such as `us-east-2`, or an Amazon Local Zone. You can use this filter with the overall traffic suggestions details query type.
- **Geo:** Specify `geo` for some queries. This is required for queries that use the `Top locations` query type, but not allowed for other query types. To understand when to specify `geo` for filter parameters, see the query examples section.

For the routing suggestions query type, you can filter the data more by specifying one or more of the following criteria:

- **Current Amazon location:** Specify an Amazon Web Services Region, such as `us-east-2`.
- **Proposed Amazon location:** Specify an Amazon Web Services Region, such as `us-east-2`, or an Amazon Local Zone.
- **IPv4 prefix:** Specify an IPv4 prefix in the standard format, similar to `192.0.2.0/24`.
- **Monitor ARN:** Specify the ARN for a specific monitor.
- **DNS resolver IP:** Specify the IP address of a DNS resolver.
- **DNS resolver ISP:** Specify the name of a DNS resolver (typically an ISP), for example, `Cloudflare`.
- **DNS resolver ASN:** Specify the autonomous system number (ASN) of a DNS resolver, for example, 4225.

The operators that you can use for filtering your data are `EQUALS` and `NOT_EQUALS`. For details about filtering parameters, see the FilterParameter API operation.

To see details about the query interface operations, see the following API operations in the Internet Monitor API Reference Guide:

- To create and run a query, see the StartQuery API operation.
- To stop a query, see the StopQuery API operation.
- To return data for a query that you've created, see the GetQueryResults API operation.
- To retrieve the status of a query, see the GetQueryStatus API operation.

Query examples

To create a query that you can use to retrieve a filtered set of data from your monitor's log file, you use the StartQuery API operation. You specify a query type and filter parameters for the query. Then, when you use the Internet Monitor query interface API operation to get query results using the query, it will retrieve the subset of your data that you want to work with.

To illustrate how query types and filter parameters work, let's look at some examples.

Example 1

Let's say that you want to retrieve all of your monitor's log file data for a specific country, except for one city. The following example shows filter parameters for a query that you could create with the `StartQuery` operation for this scenario.

```
{
   MonitorName: "TestMonitor"
   StartTime: "2023-07-12T20:00:00Z"
   EndTime: "2023-07-12T21:00:00Z"
   QueryType: "MEASUREMENTS"
   FilterParameters: [
      {
       Field: "country",
       Operator: "EQUALS",
       Values: ["Germany"]
      },
      {
       Field: "city",
       Operator: "NOT_EQUALS",
       Values: ["Berlin"]
      },
    ]
}
```

Example 2

As another example, let's say that you want to see your top locations by metropolitan area. You could use the following example query for this scenario.

```
{
   MonitorName: "TestMonitor"
   StartTime: "2023-07-12T20:00:00Z"
   EndTime: "2023-07-12T21:00:00Z"
   QueryType: "TOP_LOCATIONS"
   FilterParameters: [
      {
       Field: "geo",
       Operator: "EQUALS",
       Values: ["metro"]
      },
    ]
}
```

Example 3

Now, let's say that you want to see the top city-network combinations in the Los Angeles metro area. To do this, specify `geo=city`, and then set `metro` to Los Angeles. Now, the query returns the top city-networks in the Los Angeles metro area instead of the top metro+networks overall.

Here's the example query that you could use:

```
{
   MonitorName: "TestMonitor"
   StartTime: "2023-07-12T20:00:00Z"
   EndTime: "2023-07-12T21:00:00Z"
   QueryType: "TOP_LOCATIONS"
   FilterParameters: [
      {
       Field: "geo",
       Operator: "EQUALS",
       Values: ["city"]
      },
      {
       Field: "metro",
       Operator: "EQUALS",
       Values: ["Los Angeles"]
      }
    ]
}
```

Example 4

Next, let's say that you want to retrieve TTFB data for a specific subdivision (for example, a U.S. state).

The following is an example query for this scenario:

```
{
   MonitorName: "TestMonitor"
   StartTime: "2023-07-12T20:00:00Z"
   EndTime: "2023-07-12T21:00:00Z"
   QueryType: "TOP_LOCATION_DETAILS"
   FilterParameters: [
      {
       Field: "subdivision",
       Operator: "EQUALS",
       Values: ["California"]
      },
```

```
    ]
}
```

Example 5

Now, let's say that you want to retrieve TTFB data for every location where your application has client traffic.

The following is an example query for this scenario:

```
{
   MonitorName: "TestMonitor"
   StartTime: "2023-07-12T20:00:00Z"
   EndTime: "2023-07-12T21:00:00Z"
   QueryType: "OVERALL_TRAFFIC_SUGGESTIONS"
   FilterParameters: []
}

Results:
[us-east-1, 40, us-west-2, 30],
[us-east-1, 40, us-west-1, 35],
[us-east-1, 40, us-east-1, 44],
[us-east-1, 40, CloudFront, 22],
...
[us-east-2, 44, us-west-2, 30],
[us-east-2, 44, us-west-1, 35],
...
```

Example 6

Let's say that you want to retrieve TTFB data for a specific new Amazon Web Services Region.

The following is an example query for this scenario:

```
{
   MonitorName: "TestMonitor"
   StartTime: "2023-07-12T20:00:00Z"
   EndTime: "2023-07-12T21:00:00Z"
   QueryType: "OVERALL_TRAFFIC_SUGGESTIONS_DETAILS"
   FilterParameters: [
      {
       Field: "proposed_aws_location",
```

```
         Operator: "EQUALS",
         Values: ["us-west-2"]
        },
    ]
}

Results:
[San Jose, San Jose-Santa Clara, California, United States, 7922, us-east-1, 40, 350,
 350, us-west-2, 45]
[San Jose, San Jose-Santa Clara, California, United States, 7922, us-west-1, 35, 450,
 450, us-west-2, 45]
```

Example 7

A final example is to retrieve data for specific DNS resolvers.

The following is an example query for this scenario:

```
{
    MonitorName: "TestMonitor"
    StartTime: "2023-07-12T20:00:00Z"
    EndTime: "2023-07-12T21:00:00Z"
    QueryType: "ROUTING_SUGGESTIONS"
    FilterParameters: [
        {
         Field: "proposed_aws_location",
         Operator: "EQUALS",
         Values: ["us-east-1"]
        },
    ]
}

Results:
[162.158.180.245, 13335, Cloudflare, [5.4.0.0/14], us-east-2, 200.0, us-east-1, 160.0]
[162.158.180.243, 13313, Cloudflare, [5.4.0.0/10], us-east-2, 150.0, us-east-1, 125.0]
```

Get query results

After you define a query, you can return a set of results with the query by running another Internet Monitor API operation, GetQueryResults. When you run `GetQueryResults`, you specify the query ID for the query that you've defined, along with the name of your monitor. `GetQueryResults` retrieves data for the specified query into a result set.

When you run a query, make sure that the query has finished running before you use `GetQueryResults` to look at the results. You can determine if the query has completed by using the [GetQueryStatus](#) API operation. When the `Status` for the query is `SUCCEEDED`, you can go ahead with reviewing the results.

When your query completes, you can use the following information to help you review the results. Each query type that you use to create a query includes a unique set of data fields from the log files, as described in the following list:

Measurements

The `measurements` query type returns the following data:

`timestamp, availability, performance, bytes_in, bytes_out, rtt_p50, rtt_p90, rtt_p95`

Top locations

The `top locations` query type groups data by location, and provides the data averaged over the time period. The data that it returns includes the following:

`aws_location, city, metro, subdivision, country, asn, availability, performance, bytes_in, bytes_out, current_fbl, best_ec2, best_ec2_region, best_cf_fbl`

Note that `city`, `metro`, and `subdivision` are only returned if you choose that location type for the geo field. The following location fields are returned, depending on the location type that you specify for geo:

```
city = city, metro, subdivision, country
metro = metro, subdivision, country
subdivision = subdivision, country
country = country
```

Top locations details

The `top locations details` query type returns data grouped hour by hour. The query returns the following data:

`timestamp, current_service, current_fbl, best_ec2_fbl, best_ec2_region, best_cf_fbl`

Overall traffic suggestions

The `overall traffic suggestions` query type returns data grouped hour by hour. The query returns the following data:

`current_aws_location, proposed_aws_location, average_fbl, traffic, optimized_traffic_excluding_cf, optimized_traffic_including_cf`

Overall traffic suggestions details

The `overall traffic suggestions details` query type returns data grouped hour by hour. The query returns the following data:

`aws_location, city, metro, subdivision, country, asn, traffic, current_aws_location, fbl_data`

Routing suggestions

The `routing suggestions` query type returns data grouped hour by hour. The query returns the following data:

`dns_resolver_ip, dns_resolver_asn, dns_resolver_isp, ipv4_prefixes, current_aws_location, current_latency, proposed_aws_location, proposed_latency`

When you run the `GetQueryResults` API operation, Internet Monitor returns the following in the response:

- A *data string array* that contains the results that the query returns. The information is returned in arrays that are aligned with the `Fields` field, also returned by the API call. Using the `Fields` field, you can parse the information from the `Data` repository and then further filter or sort it for your purposes.
- An *array of fields* that lists the fields that the query returned data for (in the `Data` field response). Each item in the array is a name-datatype pair, such as `availability_score-float`.

Troubleshooting

If errors are returned when you use query interface API operations, verify that you have the required permissions to use Internet Monitor. Specifically, make sure that you have the following permissions:

```
internetmonitor:StartQuery
internetmonitor:GetQueryStatus
internetmonitor:GetQueryResults
internetmonitor:StopQuery
```

These permissions are included in the recommended Amazon Identity and Access Management policy to use the Internet Monitor dashboard in the console. For more information, see Amazon managed policies for Internet Monitor.

Add a monitor using other Amazon services

A simple way to add monitoring with Internet Monitor is to choose to create a monitor when you add a supported resource when you create a resource—or use monitoring for the resource—in console.

Resources with an integrated option for adding Internet Monitor include the following:

- VPCs
- Network Load Balancers
- Amazon CloudFront distributions

The following sections provide more information about Internet Monitor integrations in the service consoles for supported resources.

Contents

Add a monitor with a Network Load Balancer

When you create a Network Load Balancer in the Amazon Web Services Management Console, you can optionally choose to also set up monitoring for traffic to and from the Network Load Balancer using a monitor in Internet Monitor. You can add the Network Load Balancer to an existing monitor, or you can opt to create a new monitor for your Network Load Balancer traffic.

By using Internet Monitor with your Network Load Balancer, you can view and evaluate measurements and metrics about availability, performance, monitored bytes transferred, and round-trip times that are specific to your application's client locations and ASNs (typically, internet service providers). Internet Monitor also determines when there are anomalies in performance and availability, and then creates health events in your monitor, which you can choose to be notified about. To learn more about how you can use a monitor to manage and improve your clients' experience with your application, see Use a monitor in Internet Monitor.

> **Important**
>
> To create a monitor, or add a Network Load Balancer to an existing monitor, you must have the correct permissions in place. For more information, see Identity and Access Management for Internet Monitor.

Add a Network Load Balancer to an existing monitor

When you create the Network Load Balancer in the Amazon Web Services Management Console, you can choose to have Internet Monitor add the new Network Load Balancer to an existing monitor. Under **Integrations**, choose Internet Monitor, and then choose **Add monitor**. Choose **Select an existing monitor**, and then enter a monitor name. Or choose **View monitors** to go to the Internet Monitor console, and then scroll down to see a list of available monitors.

After you add the Network Load Balancer to a monitor, wait a few minutes, and then metrics for traffic to and from the load balancer will start being shown on the Internet Monitor console. To learn more about the **Status** and **Data processing status** values, see Monitoring details in Internet Monitor (Configure page).

You can edit the monitor at any time, to remove the load balancer or add another Network Load Balancer, or other resources. You can also change the percentage of traffic that you're monitoring, or make other changes. If you choose to remove the Network Load Balancer from the monitor, traffic from clients to that load balancer is no longer monitored by Internet Monitor.

To learn more about updating a monitor, see Edit a monitor in Internet Monitor.

Create a monitor for a Network Load Balancer

Under **Integrations**, choose Internet Monitor, and then choose **Monitor resource traffic**. Choose **Create a new monitor**, and then enter a monitor name. Leave the default traffic percentage to monitor, 100%, or specify a custom percentage, and then choose **Create monitor**.

After you create the monitor, wait a few minutes, and then metrics for traffic to and from the Network Load Balancer will start being shown on the Internet Monitor console. If you like, you can also choose a percentage of client traffic that you want to monitor for your application (the default is 100%).

You can learn more by reviewing the information in Step 1: Create a monitor.

Pricing

With Internet Monitor, you pay only for what you use. Pricing for Internet Monitor has two components: a per monitored resource fee and a per city-network fee. A city-network is the location that clients access your application resources from and the network (an ASN, such as an internet service provider or ISP) that clients access the resources through.

For more information, including pricing examples, see Pricing for Internet Monitor.

Stop monitoring a Network Load Balancer

If you'd like to stop monitoring your Network Load Balancer resource with Internet Monitor, do the following in the Internet Monitor console:

To remove a resource from a monitor

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the left navigation pane, under **Network Monitoring**, choose **Internet monitors**.
3. Choose your monitor, and then choose the **Action** menu.
4. Choose **Update monitor**.
5. Under **Added resources**, choose **Remove resources**.
6. Choose the Network Load Balancer to remove, and then choose **Remove**.
7. Choose **Update**.

Add an Internet Monitor monitor with Amazon VPC

When you create a Amazon Virtual Private Cloud VPC in the Amazon Web Services Management Console, you can optionally choose to also set up monitoring for it in Internet Monitor. You can add the VPC to an existing monitor, or you can opt to create a new monitor for the VPC in the Amazon VPC console.

By using Internet Monitor with your VPC, you can view and evaluate measurements and metrics about availability, performance, monitored bytes transferred, and round-trip times that are specific to your application's client locations and ASNs (typically internet service providers). Internet Monitor also determines when there are anomalies in performance and availability and creates health events in your monitor, which you can choose to be notified about. To learn more about how you can use a monitor to manage and improve your clients' experience with your application, see Use a monitor in Internet Monitor.

> **Important**
>
> To create a monitor, or add a VPC to an existing monitor, you must have the correct permissions in place. For more information, see Identity and Access Management for Internet Monitor.

Add a VPC to an existing monitor

You can choose to have Internet Monitor add a new VPC to an existing monitor for you when you create the VPC in the Amazon Web Services Management Console. After you add the VPC, wait a few minutes, and then metrics for the VPC will start being shown on the Internet Monitor console.

You can edit the monitor at any time, to remove the VPC or add another VPC or other resources. You can also change the percentage of traffic that you're monitoring, or make other changes. If you choose to remove the VPC from the monitor, traffic from clients to that VPC is no longer monitored by Internet Monitor.

To learn more about updating a monitor, see Edit a monitor in Internet Monitor.

Create a monitor for a VPC

If you opt to create a monitor for a VPC, the **Create monitor** wizard walks you through the steps. You add the VPC as a monitored resource when you create the monitor. If you like, you can also choose a percentage of client traffic that you want to monitor for your application (the default is 100%).

You can learn more by reviewing the information in Step 1: Create a monitor.

Pricing

With Internet Monitor, you pay only for what you use. Pricing for Internet Monitor has two components: a per monitored resource fee and a per city-network fee. A city-network is the location that clients access your application resources from and the network (an ASN, such as an internet service provider or ISP) that clients access the resources through.

For more information, including pricing examples, see Pricing for Internet Monitor

Stop monitoring a VPC

If you'd like to stop monitoring your VPC resource with Internet Monitor, do the following in the Internet Monitor console:

To remove a resource from a monitor

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the left navigation pane, under **Network Monitoring**, choose **Internet monitors**.
3. Choose your monitor, and then choose the **Action** menu.
4. Choose **Update monitor**.
5. Under **Added resources**, choose **Remove resources**.
6. Choose the VPC to remove, and then choose **Remove**.
7. Choose **Update**.

Add an Internet Monitor monitor with CloudFront

On the metrics dashboard for a distribution in Amazon CloudFront console, you can set up additional monitoring for a distribution in Internet Monitor. You can add the distribution to an existing monitor, or you can create a new monitor for the distribution.

By using Internet Monitor with your CloudFront distribution, you can view and evaluate measurements and metrics about availability, performance, monitored bytes transferred, and round-trip times that are specific to your application's client locations and ASNs (typically internet service providers). Internet Monitor also determines when there are anomalies in performance and availability and creates health events in your monitor, which you can choose to be notified about. To learn more about how you can use a monitor to manage and improve your clients' experience with your application, see Use a monitor in Internet Monitor.

> **Important**
>
> To create a monitor, or add a distribution to an existing monitor, you must have the correct permissions in place. For more information, see Identity and Access Management for Internet Monitor.

Add a distribution to an existing monitor

You can choose to have Internet Monitor add a distribution to an existing monitor directly from the CloudFront metrics dashboard in the Amazon Web Services Management Console. After you add the distribution, wait a few minutes, and then metrics for the distribution will start being shown on the Internet Monitor console.

You can edit the monitor at any time, to remove the distribution or add another distribution or other resources. You can also change the percentage of traffic that you're monitoring, or make other changes. If you choose to remove the distribution from the monitor, traffic from clients to that distribution is no longer monitored by Internet Monitor.

To learn more about updating a monitor, see Edit a monitor in Internet Monitor.

Create a monitor for a distribution

If you opt to create a monitor for a distribution, the **Create monitor** wizard walks you through the steps. You add the distribution as a monitored resource when you create the monitor. If you like, you can also choose a percentage of client traffic that you want to monitor for your application (the default is 100%).

You can learn more by reviewing the information in Step 1: Create a monitor.

Pricing

With Internet Monitor, you pay only for what you use. Pricing for Internet Monitor has two components: a per monitored resource fee and a per city-network fee. A city-network is the location that clients access your application resources from and the network (an ASN, such as an internet service provider or ISP) that clients access the resources through.

For more information, including pricing examples, see Pricing for Internet Monitor.

Stop monitoring a distribution

If you'd like to stop monitoring your distribution resource with Internet Monitor, do the following in the Internet Monitor console:

To remove a resource from a monitor

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the left navigation pane, under **Network Monitoring**, choose **Internet monitors**.
3. Choose your monitor, and then choose the **Action** menu.
4. Choose **Update monitor**.
5. Under **Added resources**, choose **Remove resources**.
6. Choose the distribution to remove, and then choose **Remove**.
7. Choose **Update**.

Create alarms with Internet Monitor

You can create Amazon CloudWatch alarms based on Internet Monitor metrics, just as you can for other Amazon CloudWatch metrics.

For example, you can create an alarm based on the Internet Monitor metric `PerformanceScore`, and configure it to send a notification when the metric is lower than a value that you choose. You configure alarms for Internet Monitor metrics following the same guidelines as for other CloudWatch metrics.

Following are the example Internet Monitor metrics that you might choose to create an alarm for:

- **PerformanceScore**
- **AvailabilityScore**
- **RoundtripTime**

To see all the metrics available for Internet Monitor, see [View Internet Monitor metrics or set alarms in CloudWatch Metrics](#).

The following procedure provides an example of setting an alarm on **PerformanceScore** by navigating to the metric in the CloudWatch dashboard. Then, you follow the standard CloudWatch steps to create an alarm based on a threshold that you choose, and set up a notification or choose other options.

To create an alarm for PerformanceScore in CloudWatch Metrics

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. Choose **Metrics**, and then choose **All metrics**.
3. Filter for Internet Monitor by choosing `AWS/InternetMonitor`.
4. Choose **MeasurementSource, MonitorName**.
5. In the list, select **PerformanceScore**.
6. On the **GraphedMetrics** tab, under **Actions**, choose the bell icon to create an alarm based on a static threshold.

Now, follow the standard CloudWatch steps to choose options for the alarm. For example, you can choose to be notified by an Amazon SNS message if **PerformanceScore** is below a specific threshold number. Alternatively, or in addition, you can add the alarm to a dashboard.

Keep in mind the following:

- Internet Monitor metrics are typically calculated and published within 20 minutes.
- When you create an alarm based on Internet Monitor metrics, make sure that you take into account the short delay before publication when you set an alarm's lookback period. We recommend that you configure **Evaluation Periods** with lookback period that is a minimum of 25 minutes.

To learn more about using CloudWatch alarms with Internet Monitor, see the following blog post: Using Internet Monitor for enhanced internet observability.

For more information about options when you create a CloudWatch alarm, see Create a CloudWatch alarm based on a static threshold.

Using Internet Monitor with Amazon EventBridge

Overall (global) health events that Internet Monitor creates for networking issues are published with Amazon EventBridge, so that you can send notifications about a degradation in end users' experience for your application due to a global health event.

> **Note**
>
> Local health events are not published with EventBridge.

To use EventBridge to work with Internet Monitor health events, follow the guidance here.

To set up a rule for Internet Monitor in EventBridge

1. In the Amazon Web Services Management Console, in EventBridge, choose **Rules**, then enter a name and a description. Create the rule on the **Default** event bus.
2. In Step 2, select **Other** for the event source, and then, under **Event pattern**, match the following source.

```
{
  "source": ["aws.internetmonitor"]
}
```

3. In Step 3, for the target, select **Amazon Service** and **CloudWatch Logs Group**, then select an existing log group or create a new one.
4. Add any desired tags, and then create the rule. This should populate your selected CloudWatch Logs Group with events from EventBridge.

For more information about how EventBridge rules work with event patterns, see [Amazon EventBridge event patterns] in the Amazon EventBridge User Guide.

Troubleshoot CloudWatch logs and metrics access errors

To support some features, Internet Monitor must interact with certain Amazon CloudWatch resources, including logs and metrics. If Internet Monitor can't access the CloudWatch resources that it requires access to, Internet Monitor sets a status code of `FAULT_ACCESS_CLOUDWATCH` for the monitor.

There are several reasons that your monitor might have the state `FAULT_ACCESS_CLOUDWATCH`. The following sections list possible causes for these errors, and suggested troubleshooting steps.

Internet Monitor couldn't access CloudWatch logs in your account

Internet Monitor publishes diagnostic logs about your monitored application traffic. It publishes these logs to log groups in CloudWatch Logs in the following location: `/aws/internet-monitor/monitor_name/[byCity|byMetro|bySubdivision|byCountry]`. Internet Monitor was unable to access these log groups.

Error states and potential solutions:

- **PutLogEvents throttling error:** The Internet Monitor service might have been throttled when it tried to publish your monitor's logs to CloudWatch. Review the throttling limits for your account, and, if necessary, request an increase in the limit.
- **Log group not found:** Disable, and then re-enable your monitor. Enabling a monitor restarts log group creation, which might correct the problem.
- **PutLogEvents access denied error:** Contact Amazon support for assistance.
- **PutLogEvents unknown or general error:** Contact Amazon support for assistance.

Internet Monitor couldn't access CloudWatch metrics in your account

Internet Monitor delivers specific CloudWatch metrics about the application traffic that is tracked by a monitor. An error occurred when Internet Monitor tried to deliver these metrics to CloudWatch.

Error states and potential solutions:

- **PutMetricData throttling error:** The Internet Monitor service might have been throttled when it tried to publish your monitor's metrics to CloudWatch. Review the throttling limits for your account, and, if necessary, request an increase in the limit.
- **PutMetricData access denied error:** Contact Amazon support for assistance.
- **PutMetricData unknown or general error:** Contact Amazon support for assistance.

Data protection and data privacy with Internet Monitor

The Amazon shared responsibility model applies to data protection and data privacy in Internet Monitor. As described in this model, Amazon is responsible for protecting the global infrastructure that runs all of the Amazon cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. For more information about data privacy, see the Data Privacy FAQ. For information about data protection in Europe, see The Amazon Shared Responsibility Model and GDPR blog post on the Amazon Security Blog. For more resources about complying with GDPR requirements, see the General Data Protection Regulation (GDPR) Center.

We strongly recommend that you never put sensitive identifying information, such as your end users' account numbers, email addresses, or other personal information, into free-form fields. Any data that you enter into Internet Monitor or other services might be included in diagnostic logs.

Identity and Access Management for Internet Monitor

Amazon Identity and Access Management (IAM) is an Amazon Web Services service that helps an administrator securely control access to Amazon resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use Internet Monitor resources. IAM is an Amazon Web Services service that you can use with no additional charge.

Important

Internet Monitor resource changes on July 8, 2024
If you created IAM policies that included Internet Monitor resources before July 8, 2024, be aware of the following change to Internet Monitor resources and resource types:

- Resource-level permissions for the **GetHealthEvent** action are now supported only on the **Monitor** resource type. The permissions are not supported on the **HealthEvent** resource.

To see more information about the actions, resources, and condition keys that you can specify in policies to manage access to Amazon resources in Internet Monitor, see Actions, resources, and condition keys for Internet Monitor.

Contents

Upgrade IAM policies to IPv6

Internet Monitor customers use IAM policies to set an allowed range of IP addresses, to prevent any IP addresses outside the configured range from being able to access Internet Monitor APIs.

The *internetmonitor.`region`.api.aws* endpoint, where you access Internet Monitor APIs, is being upgraded to support dual-stack (IPv4 and IPv6).

IP address filtering policies that are not updated to handle IPv6 addresses might result in clients losing access to Internet Monitor APIs.

Customers impacted by the upgrade to include IPv6

Customers who are using dual-stack with policies that contain the *aws:sourceIp* filter are impacted by this upgrade. Dual-stack means that the network supports both IPv4 and IPv6.

If you use dual-stack, you must update your IAM policies that are currently configured with IPv4 format addresses to include IPv6 format addresses.

The following summarizes recommended actions, depending on your scenario. To confirm the endpoint that your SDK uses, see Identify the Internet Monitor endpoint used by your code.

Endpoint	Using IAM policy with `aws:sourceIp` condition?	Recommended action
`internetmonitor.region.amazonaws.com` (not dual-stack)	Yes	To restrict access to IPv4 only, take no further action. Or, if you anticipate that you will need IPv6 support in the future, you can take action to ensure compatibility with both IPv4 and IPv6. To ensure future compatibility, on or after November 1, 2024, update your SDK, and then update your application to use the dual-stack endpoint by setting `useDualstackEndpoi`

Endpoint	Using IAM policy with `aws:sourceIp` condition?	Recommended action
		`nt=true` . For more information, see Dual-stack and FIPS endpoints. If you choose to use both IPv4 and IPv6, you must also update the IP address filtering condition (`aws:sourceIp`) in your IAM policies to include IPv6 addresses.
`internetmonitor.region.amazonaws.com` (not dual-stack)	No	To restrict access to IPv4 only, take no further action. Or, if you anticipate that you will need IPv6 support in the future, you can take action to ensure compatibility with both IPv4 and IPv6. To ensure future compatibility, on or after November 1, 2024, update your SDK, and then update your application to use the dual-stack endpoint by setting `useDualstackEndpoint=true` . For more information, see Dual-stack and FIPS endpoints.

Endpoint	Using IAM policy with `aws:sourceIp` condition?	Recommended action
`internetmonitor.region.api.aws`	Yes	Currently, this endpoint supports only IPV4. On November 1, 2024, IPv6 will be enabled on this endpoint. To ensure future compatibility with both IPv4 and IPv6, on or after November 1, 2024, update your SDK, and then update your application to use the dual-stack endpoint by setting `useDualstackEndpoint=true` . For more information, see [Dual-stack and FIPS endpoints](#). When you make the change to use both IPv4 and IPv6, you must also update the IP address filtering condition (`aws:sourceIp`) in your IAM policies to include IPv6 addresses. If you instead want to restrict access to IPv4 only, set `useDualstackEndpoint=false` . For more information, see [Dual-stack and FIPS endpoints](#).

Endpoint	Using IAM policy with `aws:sourceIp` condition?	Recommended action
`internetmonitor.region.api.aws`	No	Currently, this endpoint supports only IPV4. On November 1, 2024, IPv6 will be enabled on this endpoint. To ensure future compatibility with both IPv4 and IPv6, on or after November 1, 2024, update your SDK, and then update your application to use the dual-stack endpoint by setting `useDualstackEndpoint=true` . For more information, see [Dual-stack and FIPS endpoints](#). If you instead want to restrict access to IPv4 only, set `useDualstackEndpoint=false` . For more information, see [Dual-stack and FIPS endpoints](#).

For help with access issues, contact Amazon Web Services Support.

What is IPv6?

IPv6 is the next generation IP standard intended to eventually replace IPv4. IPv4 uses a 32-bit addressing scheme, to support 4.3 billion devices. IPv6 instead uses 128-bit addressing, to support approximately 340 trillion trillion trillion (or 2 to the 128th power) devices.

The following are examples of IPv6 addresses:

```
2001:cdba:0000:0000:0000:0000:3257:9652
2001:cdba:0:0:0:0:3257:9652
```

```
2001:cdba::3257:965
```

IPv6 offers a larger address space, improved routing efficiency, and better support for new internet services. By updating to dual-stack and supporting IPv6, Internet Monitor enables improved performance and scalability. Follow the steps in this section to update your configurations and take advantage of dual-stack support.

Identify the Internet Monitor endpoint used by your code

If you use an Internet Monitor SDK, start by verifying which endpoint your code is using: the IPv4 endpoint or the dual-stack (IPv4 and IPv6) endpoint. If you don't use an SDK with Internet Monitor, you can skip this section.

You can run the following code example to determine the Internet Monitor endpoint that you're using. For this example, we're using the Internet Monitor SDK for Go in the US East (N. Virginia) Region.

```
package main

import (
    "fmt"
    "log"

    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/internetmonitor"
)

func main() {
    // Create a new session with the default configuration
    sess := session.Must(session.NewSession(&aws.Config{
        Region: aws.String("us-east-1"),
    }))

    // Create a new Internet Monitor client
    internetMonitorClient := internetmonitor.New(sess)

    // Get the endpoint URL
    endpoint := internetMonitorClient.Endpoint

    fmt.Printf("Internet Monitor endpoint URL: %s\n", endpoint)
}
```

When you run this code, it returns the Internet Monitor endpoint. If you see the following response, you're using the Internet Monitor domain that supports only IPv4. You can tell because the format of the endpoint URL includes `amazonaws.com`.

```
Internet Monitor endpoint URL: https://internetmonitor.us-east-1.amazonaws.com
```

If you see the following response instead, then you're using the domain which is being upgraded to support dual-stack (IPv4 and IPv6). Here, you can tell because the endpoint URL includes `api.aws`. However, note that until the upgrade is complete, this endpoint supports only IPv4.

```
Internet Monitor endpoint URL: https://internetmonitor.us-east-1.api.aws
```

Update an IAM policy for IPv6

IAM policies use the `aws:SourceIp` filter to set an allowed range of IP addresses.

Dual-stack supports both IPv4 and IPV6 traffic. If your network uses dual-stack, you must ensure that any IAM polices that are used for IP address filtering are updated to include IPv6 address ranges.

For example, this policy allows IPv4 address ranges `192.0.2.0.*` and `203.0.113.0.*`, identified in the `Condition` element.

```
# https://docs.aws.amazon.com/IAM/latest/UserGuide/
reference_policies_examples_aws_deny-ip.html
{
    "Version": "2012-10-17",
    "Statement": {
        "Effect": "Deny",
        "Action": "*",
        "Resource": "*",
        "Condition": {
            "NotIpAddress": {
                "*aws:SourceIp*": [
                    "*192.0.2.0/24*",
                    "*203.0.113.0/24*"
                ]
            },
            "Bool": {
                "aws:ViaAWSService": "false"
```

```
            }
        }
    }
}
```

To update this policy, we'll change the policy's `Condition` element to add IPv6 address ranges, as shown in the following example:

```
"Condition": {
            "NotIpAddress": {
                "*aws:SourceIp*": [
                    "*192.0.2.0/24*", <<Existing IPv4 address - DO NOT REMOVE>>
                    "*203.0.113.0/24*", <<Existing IPv4 address  - DO NOT REMOVE>>
                    "*2001:DB8:1234:5678::/64*", <<New IPv6 IP address>>
                    "*2001:cdba:3257:8593::/64*" <<New IPv6 IP address>>
                ]
            },
            "Bool": {
                "aws:ViaAWSService": "false"
            }
        }
```

> **Important**
>
> Do not remove the existing IPv4 addresses in the policy. They are required for backward compatibility.

For more information about managing access permissions with IAM, see [Managed policies and inline policies](#) in the *Amazon Identity and Access Management User Guide*.

Test the network after updating policies

After you update your IAM policies to include support for IPv6 addresses, we recommend that you test that your network can access an IPv6 endpoint. This section provides several examples, depending on the operating system that you use.

Test network with Linux/Unix or Mac OS X

If you use Linux/Unix or Mac OS X, you can test that your network can access the IPv6 endpoint by using the following curl command.

```
curl -v -s -o /dev/null http://ipv6.ec2-reachability.amazonaws.com/
```

If you are connected over IPv6, the connected IP address displays information similar to the following:

```
* About to connect() to aws.amazon.com port 443 (#0)
*   Trying IPv6 address... connected
* Connected to aws.amazon.com (IPv6 address) port 443 (#0)
> GET / HTTP/1.1
> User-Agent: curl/7.18.1 (x86_64-unknown-linux-gnu) libcurl/7.18.1 OpenSSL/1.0.1t
 zlib/1.2.3
> Host: aws.amazon.com
```

Test network with Windows

If you use Windows, you can test that your network can access a dual-stack endpoint over IPv6 or IPv4 by using a `ping` command, such as the following:

```
ping aws.amazon.com
```

If `ping` accesses the endpoint over IPv6, the command returns IPv6 addresses.

Verify that clients can support IPv6

We recommend that before you switch to using the *internetmonitor.{region}.api.aws* endpoint, that you first verify that your clients can access other Amazon Web Services service endpoints that are already IPv6-enabled. The following steps describe how to verify this by using an existing IPv6 endpoint.

This example uses Linux and curl version 8.6.0, and uses the [Amazon Athena service](https://docs.aws.amazon.com/athena/latest/ug/what-is.html), which has IPv6-enabled endpoints located at the *api.aws* domain.

> **Note**
>
> Switch your Amazon Web Services Region to the same Region where the client is located. In this example, we use the US East (N. Virginia) – `us-east-1` endpoint.

Use the following example to verify that your clients can access an IPv6-enabled Amazon endpoint.

1. Verify that the Athena endpoint resolves with an IPv6 address by using the following command.

```
dig +short AAAA athena.us-east-1.api.aws
2600:1f18:e2f:4e05:1a8a:948e:7c08:d2d6
2600:1f18:e2f:4e03:4a1e:83b0:8823:4ce5
2600:1f18:e2f:4e04:34c3:6e9a:2b0d:dc79
```

2. Now, determine if your client network can make a connection using IPv6 by using the following command:

```
curl --ipv6 -o /dev/null --silent -w "\nremote ip: %{remote_ip}\nresponse code:
 %{response_code}\n" https://athena.us-east-1.api.aws

remote ip: 2600:1f18:e2f:4e05:1a8a:948e:7c08:d2d6
response code: 404
```

If the remote IP address was identified **and** the response code is not `0`, a network connection was successfully made to the endpoint using IPv6.

If the remote IP address is blank or the response code is `0`, the client network or the network path to the endpoint is IPv4-only. You can verify this with the following curl command:

```
curl -o /dev/null --silent -w "\nremote ip: %{remote_ip}\nresponse code:
 %{response_code}\n" https://athena.us-east-1.api.aws

remote ip: 3.210.103.49
response code: 404
```

If you run this command, and a remote IP address was identified **and** the response code is not `0`, a network connection was successfully made to the endpoint using IPv4.

How Internet Monitor works with IAM

Before you use IAM to manage access to Internet Monitor, learn what IAM features are available to use with Internet Monitor.

To see tables showing a similar high-level view of how Amazon services work with most IAM features, see Amazon services that work with IAM in the *IAM User Guide*.

IAM features you can use with Internet Monitor

IAM feature	Internet Monitor support
Identity-based policies	Yes
Resource-based policies	No
Policy actions	Yes
Policy resources	Yes
Policy condition keys (service-specific)	Yes
ACLs	No
ABAC (tags in policies)	Partial
Temporary credentials	Yes
Principal permissions	Yes
Service roles	No
Service-linked roles	Yes

Identity-based policies for Internet Monitor

Supports identity-based policies: Yes

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Define custom IAM permissions with customer managed policies in the *IAM User Guide*.

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. You can't specify the principal in an identity-based policy because it applies to the user or role to which it is attached. To learn about all of the elements that you can use in a JSON policy, see IAM JSON policy elements reference in the *IAM User Guide*.

Resource-based policies within Internet Monitor

Supports resource-based policies: No

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM role trust policies and Amazon S3 bucket policies. In services that support resource-based policies, service administrators can use them to control access to a specific resource.

Policy actions for Internet Monitor

Supports policy actions: Yes

Administrators can use Amazon JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Action` element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Policy actions usually have the same name as the associated Amazon API operation. There are some exceptions, such as *permission-only actions* that don't have a matching API operation. There are also some operations that require multiple actions in a policy. These additional actions are called *dependent actions*.

Include actions in a policy to grant permissions to perform the associated operation.

To see a list of Internet Monitor actions, see [Actions defined by Internet Monitor]() in the *Service Authorization Reference*.

Policy actions in Internet Monitor use the following prefix before the action:

```
internetmonitor
```

To specify multiple actions in a single statement, separate them with commas.

```
"Action": [
      "internetmonitor:action1",
      "internetmonitor:action2"
         ]
```

You can specify multiple actions using wildcards (*). For example, to specify all actions that begin with the word `Describe`, include the following action:

```
"Action": "internetmonitor:Describe*"
```

Policy resources for Internet Monitor

Supports policy resources: Yes

In the *Service Authorization Reference*, you can see the following information related to Internet Monitor:

- To see a list of Internet Monitor resource types and their ARNs, see Resources defined by Internet Monitor.
- To learn the actions that you can specify with the ARN of each resource, see Actions defined by Internet Monitor.

Administrators can use Amazon JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Resource` JSON policy element specifies the object or objects to which the action applies. Statements must include either a `Resource` or a `NotResource` element. As a best practice, specify a resource using its Amazon Resource Name (ARN). You can do this for actions that support a specific resource type, known as *resource-level permissions*.

For actions that don't support resource-level permissions, such as listing operations, use a wildcard (*) to indicate that the statement applies to all resources.

```
"Resource": "*"
```

Policy condition keys for Internet Monitor

Supports service-specific policy condition keys: Yes

Administrators can use Amazon JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Condition` element (or `Condition` *block*) lets you specify conditions in which a statement is in effect. The `Condition` element is optional. You can create conditional expressions that use condition operators, such as equals or less than, to match the condition in the policy with values in the request.

If you specify multiple `Condition` elements in a statement, or multiple keys in a single `Condition` element, Amazon evaluates them using a logical `AND` operation. If you specify multiple values for a single condition key, Amazon evaluates the condition using a logical `OR` operation. All of the conditions must be met before the statement's permissions are granted.

You can also use placeholder variables when you specify conditions. For example, you can grant an IAM user permission to access a resource only if it is tagged with their IAM user name. For more information, see [IAM policy elements: variables and tags] in the *IAM User Guide*.

Amazon supports global condition keys and service-specific condition keys. To see all Amazon global condition keys, see [Amazon global condition context keys] in the *IAM User Guide*.

To see a list of Internet Monitor condition keys, see [Condition keys for Internet Monitor] in the *Service Authorization Reference*. To learn with which actions and resources you can use a condition key, see [Actions defined by Internet Monitor].

ACLs in Internet Monitor

Supports ACLs: No

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

ABAC with Internet Monitor

Supports ABAC (tags in policies): Partial

Internet Monitor has *partial* support for tags in policies. It supports tagging for one resource, monitors.

To use tags with Internet Monitor, use the Amazon Command Line Interface or an Amazon SDK. Tagging for Internet Monitor is not supported with the Amazon Web Services Management Console.

To learn more about using tags in policies in general, review the following information.

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes. In Amazon, these attributes are called *tags*. You can attach tags to IAM entities (users or roles) and to many Amazon resources. Tagging entities and resources is the first step of ABAC. Then you design ABAC policies to allow operations when the principal's tag matches the tag on the resource that they are trying to access.

ABAC is helpful in environments that are growing rapidly and helps with situations where policy management becomes cumbersome.

To control access based on tags, you provide tag information in the condition element of a policy using the `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys.

If a service supports all three condition keys for every resource type, then the value is **Yes** for the service. If a service supports all three condition keys for only some resource types, then the value is **Partial**.

For more information about ABAC, see Define permissions with ABAC authorization in the *IAM User Guide*. To view a tutorial with steps for setting up ABAC, see Use attribute-based access control (ABAC) in the *IAM User Guide*.

Using temporary credentials with Internet Monitor

Supports temporary credentials: Yes

Some Amazon Web Services services don't work when you sign in using temporary credentials. For additional information, including which Amazon Web Services services work with temporary credentials, see Amazon Web Services services that work with IAM in the *IAM User Guide*.

You are using temporary credentials if you sign in to the Amazon Web Services Management Console using any method except a user name and password. For example, when you access Amazon using your company's single sign-on (SSO) link, that process automatically creates temporary credentials. You also automatically create temporary credentials when you sign in to the console as a user and then switch roles. For more information about switching roles, see Switch from a user to an IAM role (console) in the *IAM User Guide*.

You can manually create temporary credentials using the Amazon CLI or Amazon API. You can then use those temporary credentials to access Amazon. Amazon recommends that you dynamically generate temporary credentials instead of using long-term access keys. For more information, see Temporary security credentials in IAM.

Cross-service principal permissions for Internet Monitor

Supports forward access sessions (FAS): Yes

When you use an IAM user or role to perform actions in Amazon, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an Amazon Web Services service, combined with the requesting Amazon Web Services service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other Amazon Web Services services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see Forward access sessions.

Service roles for Internet Monitor

Supports service roles: No

A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Create a role to delegate permissions to an Amazon Web Services service in the *IAM User Guide*.

Service-linked role for Internet Monitor

Supports service-linked roles: Yes

A service-linked role is a type of service role that is linked to an Amazon Web Services service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your Amazon Web Services account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

For more information about the service-linked role for Internet Monitor, see Service-linked role for Internet Monitor.

For details about creating or managing service-linked roles in general in Amazon, see Amazon services that work with IAM. Find a service in the table that includes a `Yes` in the **Service-linked role** column. Choose the **Yes** link to view the service-linked role documentation for that service.

Amazon managed policies for Internet Monitor

An Amazon managed policy is a standalone policy that is created and administered by Amazon. Amazon managed policies are designed to provide permissions for many common use cases so that you can start assigning permissions to users, groups, and roles.

Keep in mind that Amazon managed policies might not grant least-privilege permissions for your specific use cases because they're available for all Amazon customers to use. We recommend that you reduce permissions further by defining customer managed policies that are specific to your use cases.

You cannot change the permissions defined in Amazon managed policies. If Amazon updates the permissions defined in an Amazon managed policy, the update affects all principal identities (users, groups, and roles) that the policy is attached to. Amazon is most likely to update an Amazon managed policy when a new Amazon Web Services service is launched or new API operations become available for existing services.

For more information, see Amazon managed policies in the *IAM User Guide*.

Amazon managed policy: CloudWatchInternetMonitorServiceRolePolicy

This policy is attached to the service-linked role named **AWSServiceRoleForInternetMonitor** to allow Internet Monitor to access resources in your account, such as Amazon Virtual Private Cloud resources or Network Load Balancers, so that you can select them when you create a monitor. For more information, see Service-linked role for Internet Monitor.

Amazon managed policy: CloudWatchInternetMonitorReadOnlyAccess

You can attach `CloudWatchInternetMonitorReadOnlyAccess` to your IAM entities. This policy grants access to read-only actions to work with monitors and data in with Internet Monitor. Attach it to IAM users and other principals who need access to only read-only actions.

Specifically, the scope of this policy includes `internetmonitor:` so that users can use read-only Internet Monitor actions and resources. It includes some `cloudwatch:` policies to retrieve information on CloudWatch metrics. It includes some `logs:` policies to manage log queries.

To view the permissions for this policy, see CloudWatchInternetMonitorReadOnlyAccess in the *Amazon Managed Policy Reference*.

Amazon managed policy: CloudWatchInternetMonitorFullAccess

You can attach `CloudWatchInternetMonitorFullAccess` to your IAM entities. This policy grants full access to Actions for Internet Monitor for working with Internet Monitor. Attach it to IAM users and other principals who need full access to Internet Monitor actions.

Specifically, scope of this policy includes `internetmonitor:` so that users can use Internet Monitor actions and resources. It includes some `cloudwatch:` policies to retrieve information on CloudWatch alarms and metrics. It includes some `logs:` policies to manage log queries. It includes some `ec2:`, `cloudfront:`, `elasticloadbalancing:`, and `workspaces:` policies to work with resources that you add to monitors so that Internet Monitor can create a traffic profile for your application. It contains some `iam:` policies to manage IAM roles.

To view the permissions for this policy, see CloudWatchInternetMonitorFullAccess in the *Amazon Managed Policy Reference*.

Internet Monitor updates to Amazon managed policies

To view details about updates to Amazon managed policies for Internet Monitor since this service began tracking these changes, see CloudWatch updates to Amazon managed policies. For automatic alerts about managed policy changes in CloudWatch, subscribe to the RSS feed on the CloudWatch Document history page.

Service-linked role for Internet Monitor

Internet Monitor uses an Amazon Identity and Access Management (IAM) service-linked role. A service-linked role is a unique type of IAM role that is linked directly to Internet Monitor. The service-linked role is predefined by Internet Monitor and includes all the permissions that the service requires to call other Amazon services on your behalf.

Internet Monitor defines the permissions of the service-linked role, and unless defined otherwise, only Internet Monitor can assume the role. The defined permissions include the trust policy and the permissions policy, and that permissions policy cannot be attached to any other IAM entity.

You can delete the role only after first deleting its related resources. This restriction protects your Internet Monitor resources because you can't inadvertently remove permissions to access the resources.

For information about other services that support service-linked roles, see Amazon services that work with IAM and look for the services that have **Yes** in the **Service-linked role** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

Service-linked role permissions for Internet Monitor

Internet Monitor uses the service-linked role named **AWSServiceRoleForInternetMonitor**. This role allows Internet Monitor to access resources in your account, such as Amazon Virtual Private Cloud resources, Amazon CloudFront distributions, Amazon WorkSpaces directories, and Network Load Balancers, so that you can select them when you create a monitor.

This service-linked role uses the managed policy `CloudWatchInternetMonitorServiceRolePolicy`.

The **AWSServiceRoleForInternetMonitor** service-linked role trusts the following service to assume the role:

- `internetmonitor.amazonaws.com`

To view the permissions for this policy, see [CloudWatchInternetMonitorServiceRolePolicy] in the *Amazon Managed Policy Reference*.

Creating a service-linked role for Internet Monitor

You do not need to manually create the service-linked role for Internet Monitor. The first time that you create a monitor, Internet Monitor creates **AWSServiceRoleForInternetMonitor** for you.

For more information, see Creating a service-linked role in the *IAM User Guide*.

Editing a service-linked role for Internet Monitor

After Internet Monitor creates a service-linked role in your account, you cannot change the name of the role because various entities might reference the role. You can edit the description of the role using IAM. For more information, see Editing a service-linked role in the *IAM User Guide*.

Deleting a service-linked role for Internet Monitor

If you no longer need to use a feature or service that requires a service-linked role, we recommend that you delete the role. That way you don’t have an unused entity that is not actively monitored or maintained. However, you must clean up the resources for the service-linked role before you can manually delete it.

After you've removed your resources from your monitors in Internet Monitor and then deleted the monitors, you can delete the service-linked role **AWSServiceRoleForInternetMonitor**.

> **Note**
>
> If the Internet Monitor service is using the role when you try to delete it, then the deletion might fail. If that happens, wait for a few minutes and then try again.

To manually delete the service-linked role using IAM

Use the IAM console, the Amazon CLI, or the Amazon API to delete the **AWSServiceRoleForInternetMonitor** service-linked role. For more information, see Deleting a service-linked role in the *IAM User Guide*.

Updates to the Internet Monitor service-linked role

For updates to **AWSServiceRoleForInternetMonitor**, the Amazon managed policy for the Internet Monitor service-linked role, see CloudWatch updates to Amazon managed policies. For automatic alerts about managed policy changes in CloudWatch, subscribe to the RSS feed on the CloudWatch Document history page.

Quotas in Internet Monitor

Internet Monitor has the following quotas.

Resource	Default quota
Monitors per Region	50
Resources per monitor	50
Days that resolved Internet Monitor health events are retained	400

Using Network Synthetic Monitor

Network Synthetic Monitor provides visibility into the performance of the network connecting your Amazon hosted applications to your on-premises destinations, and allows you to identify the source of any network performance degradation within minutes. Network Synthetic Monitor is fully managed by Amazon, and doesn't require separate agents on monitored resources. Use Network Synthetic Monitor to visualize packet loss and latency of your hybrid network connections, and set alerts and thresholds. Then, based on this information, you can take action to improve your end users’ experience.

Network Synthetic Monitor is intended for network operators and application developers who want real-time insights into network performance.

Network Synthetic Monitor key features

- Use Network Synthetic Monitor to benchmark your changing hybrid network environment with continuous real-time packet loss and latency metrics.

- When you connect by using Amazon Direct Connect, Network Synthetic Monitor can help you to rapidly diagnose network degradation within the Amazon network with the network health indicator (NHI), which Network Synthetic Monitor writes to your Amazon CloudWatch account. The NHI metric is a binary value, based on a probabilistic score about whether network degradation is within Amazon.
- Network Synthetic Monitor provides a fully-managed agent approach to monitoring, so you don't need to install agents either on VPCs or on-premises. To get started, you just need to specify a VPC subnet and an on-premises IP address. You can establish a private connection between your VPC and Network Synthetic Monitor resources by using Amazon PrivateLink. For more information, see Using CloudWatch, CloudWatch Synthetics, and CloudWatch Network Monitoring with interface VPC endpoints.
- Network Synthetic Monitor publishes metrics to CloudWatch Metrics. You can create dashboards to view your metrics, and also create actionable thresholds and alarms on the metrics that are specific to your application.

For more information, see How Network Synthetic Monitor works.

Network Synthetic Monitor terminology and components

- **Probes** — A probe is the traffic that's sent from an Amazon-hosted resource to an on-premises destination IP address. Network Synthetic Monitor metrics measured by the probe are written into your CloudWatch account for every probe that's configured in a monitor.
- **Monitor** — A monitor displays network performance and other health information for traffic that you have created Network Synthetic Monitor *probes* for. You add probes as part of creating a monitor, and then you can view network performance metrics information using the monitor. When you create a monitor for an application, you add an Amazon hosted resource as the network source. Network Synthetic Monitor then creates a list of all possible probes between the Amazon hosted resource and your destination IP addresses. You select the destinations that you want to monitor traffic for.
- **Amazon network source** — An Amazon network source is a monitor probe's originating Amazon source, which is a subnet in one of your VPCs.
- **Destination** — A destination is the target in your on-premises network for the Amazon network source. A destination is a combination of your on-premises IP addresses, network protocols, ports, and network packet size. IPv4 and IPv6 addresses are both supported.

Network Synthetic Monitor requirements and limitations

The following summarizes requirements and limitations for Network Synthetic Monitor. For specific quotas (or limits), see Network Synthetic Monitor quotas.

- Monitor subnets must be owned by the same account as the monitor.
- Network Synthetic Monitor doesn't provide automatic network failover in the event of an Amazon network issue.
- There's a charge for each probe that you create. For pricing details, see Pricing for Network Synthetic Monitor.

How Network Synthetic Monitor works

Network Synthetic Monitor is fully managed by Amazon, and doesn't require separate agents on monitored resources. Instead, you specify *probes* by providing a VPC subnet and on-premises IP addresses.

When you create a monitor in Network Synthetic Monitor for Amazon-hosted resources, Amazon creates and manages the infrastructure in the background that is required to perform round-trip time and packet loss measurements. Because Amazon manages the required configurations, you can scale your monitoring rapidly, without needing to install or uninstall agents within your Amazon infrastructure.

When probes are created, customized elastic network interfaces (ENIs) are created and attached to probe instances and customer subnets. If Network Synthetic Monitor replaces a probe instance, for example, if it becomes unhealthy, Network Synthetic Monitor detaches the ENIs and reattaches them to the probe replacement. This means that ENI IP addresses are not changed after they are created, unless you delete a probe and create a new one for the same source and destination.

Network Synthetic Monitor focuses monitoring on the routes taken by flows from your Amazon-hosted resources instead of broadly monitoring all flows from your Amazon Web Services Region. If your workloads spread across multiple Availability Zones, Network Synthetic Monitor can monitor routes from each of your private subnets.

Network Synthetic Monitor publishes round-trip time and packet loss metrics to your Amazon CloudWatch account, based on the aggregation interval that you set when you create a monitor. You can also use CloudWatch to set individual latency and packet loss thresholds for each monitor.

For example, you might create an alarm for a packet loss sensitive workload to notify you if the packet loss average is higher than a static 0.1% threshold. You can also use CloudWatch anomaly detection to alarm on packet loss or latency metrics that are outside your desired ranges.

Availability and performance measurements

Network Synthetic Monitor sends periodic active probes from your Amazon resource to your on-premises destinations. When you create a monitor, you specify the following:

- **Aggregation interval:** The time, in seconds, that CloudWatch receives the measured results. This will be either every 30 or 60 seconds. The aggregation period you choose for the monitor applies to all probes in that monitor.
- **Probe sources (Amazon resources):** A source for a probe is a VPC and associated subnets, or just a VPC subnet, in the Regions where your network operates.
- **Probe destinations (customer resources):** A destination for a probe is the combination of on-premises IP addresses, network protocols, ports, and network packet size.
- **Probe protocol:** One of the supported protocols, ICMP or TCP. For more information, see Supported communication protocols.
- **Port (for TCP):** The port that your network uses to connect.
- **Packet size (for TCP):** The size, in bytes, of each packet transmitted between your Amazon hosted resource and your destination on a single probe. You can specify a different packet size for each probe in a monitor.

A monitor publishes the following metrics:

- **Round-trip time:** This metric, measured in microseconds, is a measure of performance. It records the time it takes for the probe to be transmitted to the destination IP address and for the associated response to be received. The round-trip time is the average time observed during the aggregation interval.
- **Packet loss:** This metric measures the percentage of total packets sent and records the number of transmissions that didn't receive an associated response. No response implies that the packets were lost along the network path.

Supported communication protocols

Network Synthetic Monitor supports two protocols for probes: ICMP and TCP.

ICMP-based probes carry ICMP echo requests from your Amazon hosted resources to the destination address, and expect an ICMP echo reply in response. Network Synthetic Monitor uses the information on the ICMP echo request and reply messages to calculate round-trip time and packet loss metrics.

TCP-based probes carry TCP SYN packets from your Amazon hosted resources to the destination address and port, and expect a TCP SYN+ACK packet in response. Network Synthetic Monitor uses the information on the TCP SYN and TCP SYN+ACK messages to calculate round-trip time and packet loss metrics. Network Synthetic Monitor periodically switches source TCP ports to increase network coverage, which increases the probability of detecting packet loss.

Network health indicator for Amazon

Network Synthetic Monitor publishes a network health indicator (NHI) metric that provides information on issues with the Amazon network for paths that include destinations connected through Amazon Direct Connect. NHI is not supported for Amazon Direct Connect attachments that are created using Cloud WAN or that use Amazon Transit Gateway. The NHI binary value is based on a statistical measure of the health of the Amazon-controlled network path from the Amazon hosted resource, where the monitor is deployed, to the Direct Connect location.

Network Synthetic Monitor uses anomaly detection to calculate availability drops or lower performance along the network paths.

> **Note**
>
> Each time that you create a new monitor, add a probe, or re-activate a probe, the NHI for the monitor is delayed by a few hours while Amazon collects data to perform anomaly detection.

To provide the NHI value, Network Synthetic Monitor applies statistical correlation across Amazon sample datasets, as well as to the packet loss and round-trip latency metrics for traffic simulating your network path. NHI can be one of two values: 1 or 0. A value of 1 indicates that Network Synthetic Monitor observed a network degradation within the Amazon controlled network path. A value of 0 indicates that Network Synthetic Monitor did not observe any network degradation for the Amazon network along the path. Using the NHI value enables you to more quickly gain awareness of the cause of network issues. For example, you can set alerts on the NHI metric so you're notified about ongoing issues with the Amazon network along your network paths.

Support for IPv4 and IPv6 addresses

Network Synthetic Monitor provides availability and performance metrics over IPv4 or IPv6 networks, and can monitor either IPv4 or IPv6 addresses from dual-stack VPCs. Network Synthetic Monitor doesn't allow both IPv4 and IPv6 destinations to be configured in the same monitor; you can create separate monitors for IPv4-only and IPv6-only destinations.

Supported Amazon Web Services Regions for Network Synthetic Monitor

The Amazon Web Services Regions where Network Synthetic Monitor is supported are listed in this section. For more information about Regions that Network Synthetic Monitor is supported in, including opt-in Regions, see Network Synthetic Monitor endpoints and quotas in the *Amazon Web Services General Reference*.

Region name	Region
Asia Pacific (Hong Kong)	ap-east-1
Asia Pacific (Mumbai)	ap-south-1
Asia Pacific (Seoul)	ap-northeast-2
Asia Pacific (Singapore)	ap-southeast-1
Asia Pacific (Sydney)	ap-southeast-2
Asia Pacific (Tokyo)	ap-northeast-1
Canada (Central)	ca-central-1
Europe (Frankfurt)	eu-central-1
Europe (Ireland)	eu-west-1
Europe (London)	eu-west-2
Europe (Paris)	eu-west-3
Europe (Stockholm)	eu-north-1

Region name	Region
Middle East (Bahrain)	me-south-1
South America (São Paulo)	sa-east-1
US East (N. Virginia)	us-east-1
US East (Ohio)	us-east-2
US West (N. California)	us-west-1
US West (Oregon)	us-west-2

Pricing for Network Synthetic Monitor

With Network Synthetic Monitor, there are no upfront costs or long-term commitments. Pricing for Network Synthetic Monitor has the following two components:

- An hourly fee per monitored Amazon resource
- CloudWatch metrics fees

When you create a monitor in Network Synthetic Monitor, you associate Amazon resources (sources) with it to be monitored. For Network Synthetic Monitor, these resources are subnets in Amazon Virtual Private Cloud (VPC). For each resource, you can create up to four probes, each of which is for traffic from a subnet in the VPC to four of your destination IP addresses. To help control your bill, you can adjust your subnet coverage and on-premises IP address destination coverage by reducing the number of resources that you monitor.

For more information about pricing, see the Amazon CloudWatch pricing page.

Network Synthetic Monitor API operations

The following table lists Network Synthetic Monitor API operations that you can use with Amazon CloudWatch. Refer to this table for links to relevant documentation.

Action	API reference	More information
Create a monitor between a source subnet and destination IP address.	See CreateMonitor	See Create a monitor
Create a probe within a monitor.	See CreateProbe	See Activate or deactivate a probe
Remove a monitor.	See DeleteMonitor	See Delete a monitor
Delete a specific probe.	See DeleteProbe	See Delete a probe
Get information about a monitor.	See GetMonitor	See Working with monitors and probes in Network Synthetic Monitor
Get information about a specific probe.	See GetProbe	See Working with monitors and probes in Network Synthetic Monitor
Get a list of all your monitors.	See ListMonitors	See Working with monitors and probes in Network Synthetic Monitor
List tags assigned to a resource.	See ListTagsForResource	See Tag or untag resources
Add key-value pairs, or tags, to a monitor or probe.	See TagResource	See Tag or untag resources
Remove a key-value pair, or tag, from a monitor or probe.	See UntagResource	See Tag or untag resources
Update an aggregation period for a monitor.	See UpdateMonitor	See Edit a monitor
Update a probe in a monitor.	See UpdateProbe	See Edit a probe

Working with monitors and probes in Network Synthetic Monitor

To get started, create a monitor with probes in Network Synthetic Monitor to measure network performance over a specified aggregation period. Then, you can update a monitor to make desired changes, for example, to change the aggregation period, deactivate or activate probes, or add or remove tags.

The following sections provide step-by-step instructions for completing these tasks for your monitors and probes by using the Amazon CloudWatch console. You can also make changes to your monitor by using the Amazon Command Line Interface.

Topics

Create a monitor

The following sections describe how to create a monitor in Network Synthetic Monitor, including the required probes. When you create a monitor, you specify probes by choosing source subnets, and then adding up to four destinations for each one. Each source-destination pair is a probe.

You can make changes to a monitor after you create it, for example, to add, remove, or deactivate probes. For more information, see Working with monitors and probes in Network Synthetic Monitor.

You can work with monitors and probes by using either the Amazon CloudWatch console or the Amazon Command Line Interface. To work with Network Synthetic Monitor programmatically, see the Network Synthetic Monitor API Reference and networkmonitor in the Amazon Command Line Interface Command Reference.

The following procedures provide step-by-step instructions for how to create a monitor by using the Amazon CloudWatch console.

- Define monitor details
- Choose sources and destinations
- Confirm probes
- Review and create monitor

> **Important**
>
> These steps are designed to be completed all at once. You can't save in-process work to continue later.

Define monitor details

The first step to create a monitor is to define the basic details, by giving the monitor a name and defining the aggregation period. Optionally, you can also add tags.

To define monitor details

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/, and then, under **Network Monitoring**, choose **Synthetic monitors**.
2. Choose **Create monitor**.
3. For **Monitor name**, enter a name for the monitor.
4. For **Aggregation period**, choose how often you want to send metrics to CloudWatch: **30 seconds** or **60 seconds**.

   > **Note**
   >
   > A shorter aggregation period provides faster detection of network issues. However, the aggregation period that you choose can affect your billing costs. For more information about pricing, see the Amazon CloudWatch pricing page.

5. (Optional) For **Tags**, add **Key** and **Value** pairs to help identify this resource, so that you can search or filter on specific information.

1. Choose **Add new tag**.
2. Enter a **Key** name and associated **Value**.
3. Choose **Add new tag** to add the new tag.

   You can add multiple tags by choosing **Add new tag**, or you can remove a tag by choosing **Remove**.
4. If you want to associate your tags with the probes for the monitor, keep **Add tags to probes created by monitor** selected. This adds the tags to the monitor probes, which can be helpful for tag-based authentication or metering.

6. Choose **Next**. On the next page, you'll specify sources and destinations, to create probes for the monitor.

Choose sources and destinations

For each monitor in Network Synthetic Monitor, you specify one or more probes, which are a combination of an Amazon source and a destination.

- A source for a probe is a VPC and associated subnets (or just a VPC subnet) in the Regions where your network operates.
- A destination is the combination of on-premises IP addresses, network protocols, ports, and network packet size.

> **Important**
>
> These steps are designed to be completed all at once. You can't save in-process work to continue later.

To choose sources and destinations

1. Prerequisite: Define monitor details.
2. Under **Amazon network source**, choose one or more subnets to include in the monitor. To choose all subnets within a VPC, choose the VPC. Or, choose specific subnets within a VPC. The subnets that you choose are the monitor sources.

3. For **Destination 1**, enter a destination IP address of the on-premises network. IPv4 and IPv6 addresses are both supported.
4. Choose **Advanced settings**.
5. For **Protocol**, choose the network protocol for the on-premises destination. The protocol can be either **ICMP** or **TCP**.
6. If you choose **TCP**, enter the following information:
    1. Enter the **Port** that your network uses to connect. The port must be a number from **1** to **65535**.
    2. Enter the **Packet size**. This is the size, in bytes, of each packet that's sent on the probe, between the source and destination. Packet size must be a number from **56** to **8500**.
7. Choose **Add destination** to add another on-premises destination to the monitor. Repeat these steps for each destination that you want to add.
8. When you're finished adding sources and destinations, choose **Next** to confirm the probes for the monitor.

Confirm probes

On the **Confirm probes** page, review all the probes that will be created for the monitor, to make sure that they're the correct set of sources and destinations.

The **Confirm probes** page shows all the possible combinations of the sources and destinations for the probe specifications that you provided in the previous step. For example, if you have six source subnets and four destination IP addresses, there are 24 possible probe combinations, so 24 probes will be created.

> **Important**
>
> - These steps are intended to be completed in one session. You can't save in-process work to continue later.
> - The **Confirm probes** page does not indicate whether a probe is valid. We recommend that you review this page carefully, and then delete any probes that aren't valid. You might be charged for probes that aren't valid if you don't remove them.

To confirm monitor probes

1. Prerequisite: Choose sources and destinations.
2. On the **Confirm probes** page, review the list of source and destination probe combinations.
3. Choose any probes that you want to remove from the monitor, and then choose **Remove**.

> **Note**
>
> You are not prompted to confirm deleting a probe. If you delete a probe and want to restore it, you must set it up again. You can add a probe to an existing monitor by following the steps in Add a probe to a monitor.

4. Choose **Next**, and then review the monitor details.

Review and create monitor

The final step is to review the details of the monitor and the probes for the monitor, and then create the monitor. You can change any information about the monitor at this point.

When you have finished reviewing and updating any information that isn't correct, create the monitor.

As soon as you create the monitor, Network Synthetic Monitor begins tracking metrics and you'll start being charged for probes in the monitor.

> **Important**
>
> - This step is intended to be completed in one session. You can't save in-process work to continue later.
> - If you choose to edit a section, you must step through the process to create a monitor from the point that you make the edits. Earlier monitor creation pages maintain the information that you already entered.

To review and create a monitor

1. On the **Review and create probes** page, choose **Edit** for any section where you want to make changes.

2. Make changes in that section, and then choose **Next**.
3. When you're finished making edits, choose **Create monitor**.

   The Network Synthetic Monitor page displays the current state of monitor creation in the **Monitors** section. When Network Synthetic Monitor is still creating the monitor, the **State** is **Pending**. When the **State** changes to **Active**, you can view CloudWatch metrics in the monitor dashboard.

   For information on working with the monitor dashboard, see Network Synthetic Monitor dashboards.

> **Note**
>
> It can take several minutes for a newly-added monitor to begin collecting network metrics.

Edit a monitor

You can edit information for a Network Synthetic Monitor, including change the name, setting a new aggregation period, or adding or removing tags. Changing a monitor's information does not change any of its associated probes.

You can work with monitors and probes by using either the Amazon CloudWatch console or the Amazon Command Line Interface. To work with Network Synthetic Monitor programmatically, see the Network Synthetic Monitor API Reference and networkmonitor in the Amazon Command Line Interface Command Reference.

To edit a monitor using the console

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/, and then, under **Network Monitoring**, choose **Synthetic monitors**.
2. In the **Monitors** section, choose the monitor that you want to edit.
3. On the monitor dashboard page, choose **Edit**.
4. For the **Monitor name**, enter the new name for the monitor.
5. For the **Aggregation period**, choose how often you want to send metrics to CloudWatch. Valid periods are:

   - **30 seconds**

- **60 seconds**

> **Note**
>
> A shorter aggregation period provides faster detection of network issues. However, the aggregation period that you choose can affect your billing costs. For more information about pricing, see the [Amazon CloudWatch pricing] page.

6. (Optional) In the **Tags** section, add **Key** and **Value** pairs to further help identify this resource, allowing you to search or filter on specific information. You can also just change the **Value** of any current **Key**.

    1. Choose **Add new tag**.
    2. Enter a **Key** name and associated **Value**.
    3. Choose **Add new tag** to add the new tag.

        You can add multiple tags by choosing **Add new tag**, or you can remove a tag by choosing **Remove**.
    4. If you want to associate your tags with the monitor, keep **Add tags to probes created by monitor** checked. This adds the tags to the monitor probes, which can be helpful if you're using tag-based authentication or metering.

7. Choose **Save changes**.

Delete a monitor

Before you can delete a monitor in Network Synthetic Monitor, you must deactivate or delete all probes associated with the monitor, regardless of the monitor **State**. After a monitor is deleted, you are no longer be charged for probes in the monitor. Be aware that you can't restore a deleted monitor.

You can work with monitors and probes by using either the Amazon CloudWatch console or the Amazon Command Line Interface. To work with Network Synthetic Monitor programmatically, see the Network Synthetic Monitor API Reference and networkmonitor in the Amazon Command Line Interface Command Reference.

To delete a monitor using the console

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/, and then, under **Network Monitoring**, choose **Synthetic monitors**.
2. In the **Monitors** section, choose the monitor that you want to delete.
3. Choose **Actions**, and then choose **Delete**.
4. If you have any active probes for the monitor, you're prompted to deactivate them. Choose **Deactivate probes**.

   > **Note**
   >
   > You can't cancel or undo this action after you choose **Deactivate probes**. Deactivated probes, however, aren't removed from the monitor. If you like, you can reactivate them later. For more information, see Activate or deactivate a probe.

5. Enter **`confirm`** in the confirmation field, and then choose **Delete**.

Alternatively, you can delete a monitor programmatically, for example, by using the Amazon Command Line Interface.

To delete a monitor by using the CLI

1. To delete a monitor, you need the monitor name. If you don't know the name, get a list of your monitors by running the list-monitors command. Note the name of the monitor that you want to delete.
2. Verify whether the monitor contains any active probes. Use get-monitor with the monitor name from the previous step. This returns a list of any probes associated with that monitor.
3. If the monitor contains active probes, you must first either set the probes to inactive or delete them.
   - To set a probe to inactive, use update-probe, and set the state to `INACTIVE`.
   - To delete a probe, use delete-probe.
4. After the probes are set to `INACTIVE` or deleted, you can delete the monitor by running the delete-monitor command. Inactive probes are not deleted when you delete the monitor.

Activate or deactivate a probe

You can activate or deactivate a probe in a monitor in Network Synthetic Monitor. You might want to deactivate a probe, for example, if you aren't currently using it but might want to use it again in the future. By deactivating a probe instead of deleting it, you won't need to spend time setting it up again. You are not billed for deactivated probes.

You can work with monitors and probes by using either the Amazon CloudWatch console or the Amazon Command Line Interface. To work with Network Synthetic Monitor programmatically, see the Network Synthetic Monitor API Reference and networkmonitor in the Amazon Command Line Interface Command Reference.

To set a probe to active or inactive by using the console

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/, and then, under **Network Monitoring**, choose **Synthetic monitors**.
2. Choose the **Monitor details** tab.
3. In the **Probes** section, choose the probe that you want to activate or deactivate.
4. Choose **Actions**, and then choose **Activate** or **Deactivate**.

> **Note**
>
> When you reactivate a probe, you begin incurring billing charges on the probe again.

Add a probe to a monitor

You can add a probe to an existing monitor in Network Synthetic Monitor. Note that when you add probes to a monitor, your billing structure is updated to include the new probe.

You can work with monitors and probes by using either the Amazon CloudWatch console or the Amazon Command Line Interface. To work with Network Synthetic Monitor programmatically, see the Network Synthetic Monitor API Reference and networkmonitor in the Amazon Command Line Interface Command Reference.

To add a probe to a monitor using the console

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/, and then, under **Network Monitoring**, choose **Synthetic monitors**.

2. In the **Monitors** section, do one of the following:

   - Choose the **Name** link of the monitor that you want to add a probe to. Choose the **Monitor details** tab, and then in the **Probes** section, choose **Add probe**.
   - Choose the monitor check box, choose **Actions**, and then choose **Add probe**.

3. On the **Add probe** page, do the following:

   1. Under **Amazon network source**, choose a subnet to add to the monitor.

      > **Note**
      >
      > You can only add one probe at a time and up to four probes per monitor.

   2. Enter the destination **IP address** of the on-premises network. Both IPv4 and IPv6 addresses are supported.
   3. Choose **Advanced settings**.
   4. Choose the network **Protocol** for the destination. It can be either **ICMP** or **TCP**.
   5. If the **Protocol** is **TCP**, enter the following information. Otherwise, skip to the next step:
      - Enter the **Port** that your network uses to connect. The port must be a number from **1** to **65535**.
      - Enter the **Packet size**. This is the size, in bytes, of each packet sent along the probe between the source and destination. Packet size must be a number from **56** to **8500**.

4. (Optional) In the **Tags** section, add **Key** and **Value** pairs to further help identify this resource, allowing you to search or filter on specific information.

   1. Choose **Add new tag**.
   2. Enter a **Key** name and associated **Value**.
   3. Choose **Add new tag** to add the new tag.

      You can add multiple tags by choosing **Add new tag**, or you can remove any tag by choosing **Remove**.

5. Choose **Add probe**.

   While the probe is being activated, the **State** shows **Pending**. It might take several minutes for the probe to become **Active**.

Edit a probe

You can change any information for an existing probe, regardless of whether that probe is active or inactive.

You can work with monitors and probes by using either the Amazon CloudWatch console or the Amazon Command Line Interface. To work with Network Synthetic Monitor programmatically, see the Network Synthetic Monitor API Reference and networkmonitor in the Amazon Command Line Interface Command Reference.

To edit a probe by using the console

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/, and then, under **Network Monitoring**, choose **Synthetic monitors**.

   Under **Name**, choose a monitor link to open the monitor dashboard.
2. Choose the **Monitor details** tab.
3. In the **Probes** section, choose the link for the probe that you want to edit.
4. On the probe details page, choose **Edit**.
5. On the **Edit *probe*** page, enter the new destination **IP address** for the probe. IPv4 and IPv6 addresses are both supported.
6. Choose **Advanced settings**.
7. Choose a network **Protocol**, **ICMP** or **TCP**.
8. If the **Protocol** is **TCP**, enter the following information:

   - Enter the **Port** that your network uses to connect. The port must be a number from **1** to **65535**.
   - Enter the **Packet size**. This is the size, in bytes, of each packet sent along the probe between the source and destination. Packet size must be a number from **56** to **8500**.
9. (Optional) Add, change, or remove Tags for the probe.
10. Choose **Save changes**.

Delete a probe

You can delete a probe rather than deactivating it if you know that you won't need it again later. You can't recover a deleted probe; instead, you must recreate it. Billing charges end for a probe when the probe is deleted.

You can work with monitors and probes by using either the Amazon CloudWatch console or the Amazon Command Line Interface. To work with Network Synthetic Monitor programmatically, see the Network Synthetic Monitor API Reference and networkmonitor in the Amazon Command Line Interface Command Reference.

To delete a probe using the console

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/, and then, under **Network Monitoring**, choose **Synthetic monitors**.
2. In the **Monitors** section, under **Name**, choose a monitor link to open the monitor dashboard.
3. Choose the **Monitor details** tab.
4. Choose the monitor check box, choose **Actions**, and then choose **Delete**.
5. In the **Delete probe** dialog box, do the following:
6. Choose **Delete** to confirm that you want to delete the probe.

   The **State** of the probe in the **Probes** section shows **Deleting**. After it's deleted, the probe is removed from the **Probes** section.

Tag or untag resources

You can work with resource tags in Network Synthetic Monitor, to add or remove tags.

You can update tags by updating monitors or probes in the console. Or, you can work with tags programmatically, for example, by using the Amazon Command Line Interface.

To update monitor tags by using the CLI

- To list resource tags, use list-tags-for-resources.
- To tag a resource, use tag-resource.
- To untag a resource, use untag-resource.

Network Synthetic Monitor dashboards

You can use dashboards in Network Synthetic Monitor to determine if a network issue is caused by Amazon, by using the network health indicator (NHI), and view probe round-trip time and packet loss. You can view this information and metrics for monitors, as well as for individual probes.

Network Synthetic Monitor creates several metrics that you can view in CloudWatch Metrics. You can specify alarms for the metrics that Network Synthetic Monitor returns. For more information, see Probe alarms.

Topics

Monitor dashboards

You can use the monitor dashboard in Network Synthetic Monitor to view the network health indicator (NHI), as well as probe round-trip time and packet loss at a monitor level. That is, a monitor dashboard shows this information for all the probes created for the monitor.

Network Synthetic Monitor also has dashboards for probes, to view the information at a probe level. For more information, see Probe dashboards.

To access a monitor dashboard

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/, and then under **Network Monitoring**, choose **Synthetic monitors**.
2. In the **Monitors** section, choose the **Name** link to open the monitor dashboard.

Overview page

The **Overview** page displays the following information for a monitor:

- **Network health** — Network health displays the network health indicator (NHI) value, which pertains to health of only the Amazon network. The NHI status is displayed as **Healthy** or **Degraded**. A **Healthy** status indicates that Network Synthetic Monitor did not observe issues with the Amazon network. A **Degraded** status indicates that Network Synthetic Monitor observed an issue with the Amazon network. The status bar in this section shows the status of the network health indicator over a default time of one hour. Hover over any point in the status bar to view additional details.

- **Probe traffic summary** — Displays the current state of the traffic between the source Amazon subnets specified for the probes in the monitor and the probes' destination IP addresses. This summary displays the following:
  - **Probes in alarm** — This number indicates how many of your probes in this monitor are in a degraded state. An alarm is triggered when a metric that you've set up as an alarm is triggered. For information on creating alarms for metrics in Network Synthetic Monitor, see Probe alarms.
  - **Packet loss** — The number of packets that were lost from the source subnet to the destination IP address. This is represented as a percentage of the total packets sent.
  - **Round-trip time** — The time it takes, displayed in milliseconds, for a packet from the source subnet to reach the destination IP address, and then come back again. Round-trip time is the average RTT observed during the aggregation period.

The data is represented on an interactive graph, so you can explore to learn details.

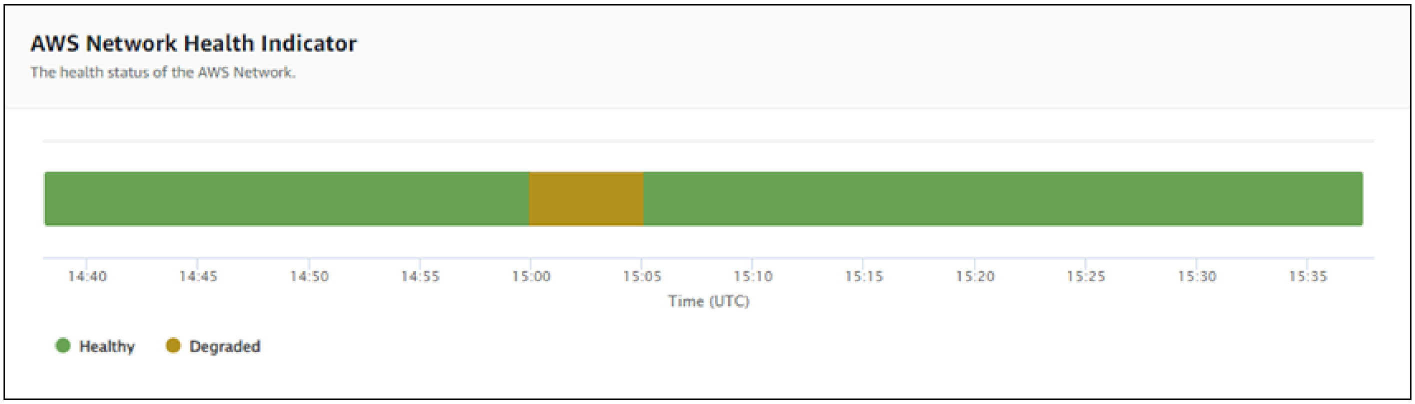
By default, data is displayed for a two-hour time frame, calculated from the current date and time. However, you can change the range to fit your needs. For more information, see Specify metrics time frame.

Tracking metrics

The **Overview** dashboard in Network Synthetic Monitor displays a graphical representation of a monitor and probes. The following graphs are shown:

- **Network health indicator** — This represents the the NHI values over a specified period. NHI indicates whether a network issue is due to problems with the Amazon network. NHI is displayed as **Healthy** (no issue with the Amazon network) or **Degraded** (there is an issue with the Amazon network).

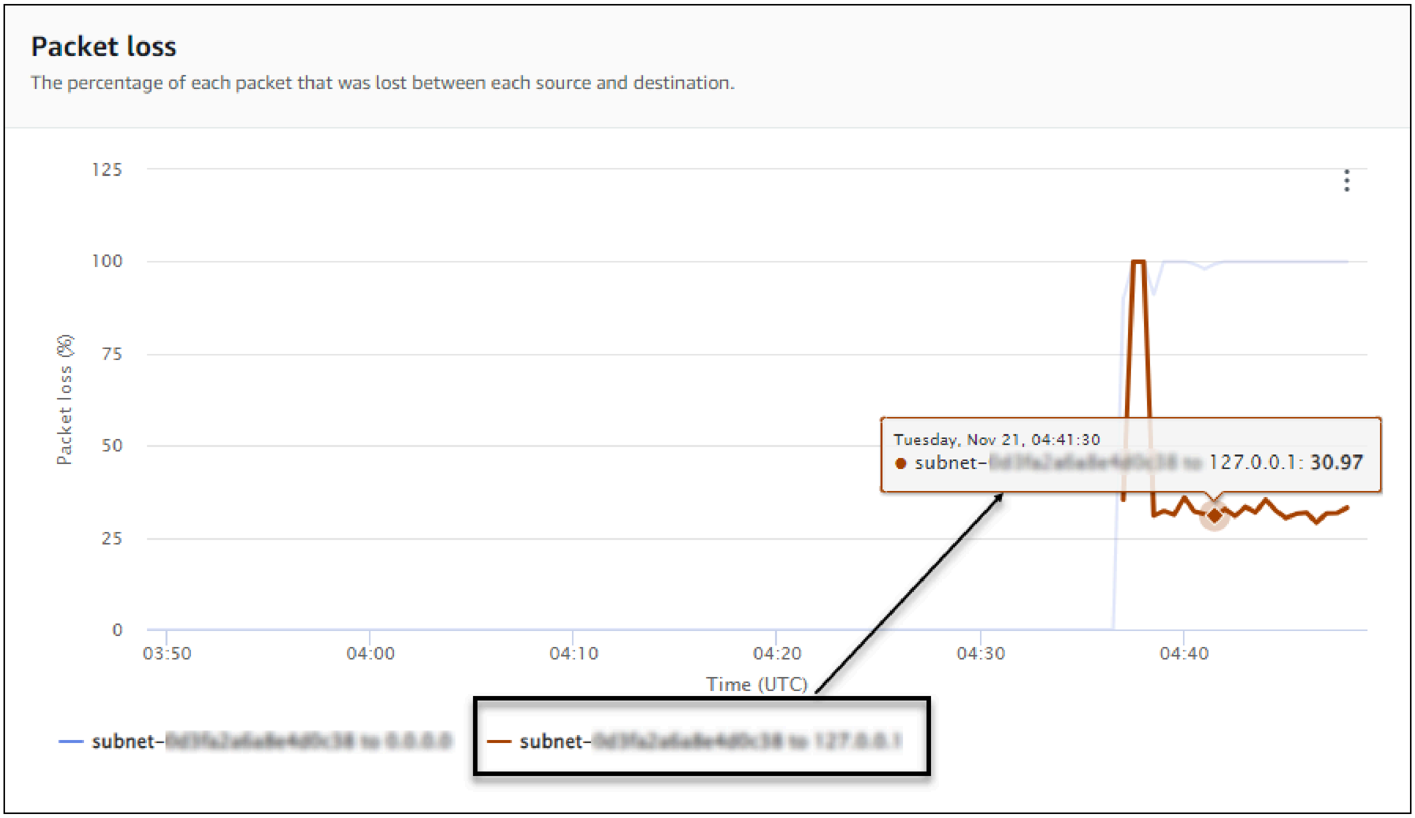
  In the following example, you can see that from 15:00 UTC until 15:05 UTC, there was a network issue that was due to an Amazon network issue **(Degraded**). After 15:05, the network issue with the Amazon network ended, so the value returned to **Healthy**. You can hover over any section of the graph to see additional details.



> **Note**
>
> The NHI indicates that an issue is due to the Amazon network. It does not describe the overall health of the Amazon network nor the health of Network Synthetic Monitor probes.

- **Packet loss** — This graph displays a line that shows the percentage of packet loss for each probe in a monitor. The legend at the bottom of the page displays each of the probes in the monitor, color-coded for uniqueness. You can hover over a probe in the chart to see the source subnet, the destination IP address, and the percentage of packet loss.

  In the following example, a packet loss alarm was created for a probe from a subnet to IP address 127.0.0.1. The alarm was triggered when the packet loss threshold was exceeded for the probe. If you hover over the graph, you can see the probe source and destination, and that there was a 30.97% packet loss for this probe on November 21 at 02:41:30.



- **Round-trip time** — This graph displays a line that shows the round-trip time for each probe. The legend at the bottom of the page displays each of the probes in the monitor, color-coded for uniqueness. You can hover over a probe in the chart to see the source subnet, the destination IP address, and the round-trip time.

  The following example shows that on Tuesday, Nov 21, at 21:45:30, the round-trip time for a probe from a subnet to IP address 127.0.0.1 was 0.075 seconds.

Monitor details

The **Monitor details** page displays details about your monitor, including a list of the probes for the monitor. You can update or add tags, or add a probe. The page includes the following sections:

- **Monitor details** — This page provides details about your monitor. You can't edit information in this section. However, you can view details of the Network Synthetic Monitor service-linked role: choose the **Role name** link to see details.
- **Probes** — This section displays a list of all probes associated with the monitor. Choose a **VPC** or **Subnet ID** link to open the VPC or subnet details in the Amazon VPC Console. You can modify a probe, to activate or deactivate it. For more information, see Working with monitors and probes in Network Synthetic Monitor.

  The **Probes** section displays information about each probe set up for that monitor, including the probe **ID**, the **VPC ID**, the **Subnet ID**, **IP address**, **Protocol**, and whether the probe is **Active** or **Inactive**.

  If you've created an alarm for a probe, the current **Status** of the alarm is shown. A status of **OK** indicates that there are no metrics events have triggered any alarms. A status of **In alarm** indicates that a metric that you created in CloudWatch triggered an alarm. If no status is displayed for a probe, then there isn't a CloudWatch alarm for it. For information on the types of Network Synthetic Monitor probe alarms that you can create, see Probe alarms.
- **Tags** — View the current tags for a monitor. You can add or remove tags by choosing **Manage tags**. This opens the **Edit probe** page. For more information on editing tags, see Edit a monitor.

Probe dashboards

You can use a **Probe** dashboard in Network Synthetic Monitor to view the network health indicator (NHI) for a probe, as well as information about round-trip time and packet loss for specific probes. There are two dashboards for probes: an **Overview** page and **Probe details** page.

You can create CloudWatch alarms to set packet loss and round-trip time metric thresholds. When a threshold is reached for a metric, a CloudWatch alarm notifies you. For information on creating probe alarms, see Probe alarms.

To access a probe dashboard

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/, and then, under **Network Monitoring**, choose **Synthetic monitors**.

2. In the **Monitors** section, choose the **Name** link to open the dashboard for a specific monitor.
3. To view the dashboard for a specific probe, choose the **ID** link for the probe.

Overview page

The **Overview** page displays the following information for a probe:

- **Network health** — Network health displays the network health indicator (NHI) value, which pertains only to health of the Amazon network. The NHI status is provided as **Healthy** or **Degraded**. A **Healthy** status indicates that Network Synthetic Monitor did not observe issues with the Amazon network for a probe. A **Degraded** status indicates that Network Synthetic Monitor observed an issue with the Amazon network. The status bar in this section shows the status of the network health indicator over a default time of one hour. Hover over any point in the status bar to view additional details.
- **Packet loss** — The number of packets that were lost from the source subnet to the destination IP address for this probe.
- **Round-trip time** — The time it takes, in milliseconds, for a packet from the source subnet to reach the destination IP address, and then come back again. Round-trip time (RTT) is the average RTT observed during the aggregation period.

Probe details

The **Probe details** page displays information about a probe, including the source and destination. You can also edit the probe, for example, to activate or deactivate it. For more information, see Working with monitors and probes in Network Synthetic Monitor.

- **Probe details** — This section provides general information about the probe, which can't be edited.
- **Probe source and destination** — This section displays details about the probe. Choose a **VPC** or **Subnet ID** link to open the VPC or subnet details in the Amazon VPC Console. You can modify a probe, for example, to activate or deactivate it.
- **Tags** — View the current tags for a monitor. You can add or remove tags by choosing **Manage tags**. This opens the **Edit probe** page. For more information on editing tags, see Edit a probe.

Specify metrics time frame

Metrics and events on the dashboards in Network Synthetic Monitor use a default time of two hours, calculated from the current time, but you can set a custom metrics default time frame to use. You can change the default to one of the following presets for the metrics time frame:

- **1h** — one hour
- **2h** — two hours
- **1d** — one day
- **1w** — one week

You can also set a custom time frame. Choose **Custom**, choose an **Absolute** or **Relative** time, and then set the time frame to a time of your own choosing. Relative time supports only 15 days back from today's date, following CloudWatch guidelines.

Additionally, you can choose the time displayed in the charts to be based on either the UTC time zone or a local time zone.

For more information, see Changing the time range or time zone format of a CloudWatch dashboard.

Probe alarms

You can create Amazon CloudWatch alarms based on Network Synthetic Monitor metrics, just as you can for other Amazon CloudWatch metrics. Any alarm that you create will appear in the probe's **Status** column of the **Monitor details** section of the Network Synthetic Monitor dashboard when the alarm is triggered. The status will either be **OK** or **In Alarm**. If no status displays for a probe, then no alarm was created for that probe.

For example, you can create an alarm based on the Network Synthetic Monitor metric `PacketLoss`, and configure it to send a notification when the metric is higher than a value that you choose. You configure alarms for Network Synthetic Monitor metrics following the same guidelines as for other CloudWatch metrics.

The following metrics are available under `AWS/NetworkMonitor` when creating a CloudWatch alarm for Network Synthetic Monitor.

- **HealthIndicator**

- **PacketLoss**
- **RTT (Round-trip time)**

For the steps to create a Network Synthetic Monitor alarm in CloudWatch, see Create a CloudWatch alarm based on a static threshold.

Data security and data protection in Network Synthetic Monitor

Cloud security at Amazon is the highest priority. As an Amazon customer, you benefit from data centers and network architectures that are built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between Amazon and you. The shared responsibility model describes this as security *of* the cloud and security *in* the cloud:

- **Security of the cloud** – Amazon is responsible for protecting the infrastructure that runs Amazon services in the Amazon Web Services Cloud. Amazon also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the Amazon Compliance Programs. To learn about the compliance programs that apply to Network Synthetic Monitor, see Amazon Services in Scope by Compliance Program.
- **Security in the cloud** – Your responsibility is determined by the Amazon service that you use. You are also responsible for other factors including the sensitivity of your data, your company's requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using Network Synthetic Monitor. The following topics show you how to configure Network Synthetic Monitor to meet your security and compliance objectives. You also learn how to use other Amazon services that help you to monitor and secure your Network Synthetic Monitor resources.

Topics

Data protection in Network Synthetic Monitor

The Amazon shared responsibility model applies to data protection in Network Synthetic Monitor. As described in this model, Amazon is responsible for protecting the global infrastructure that runs all of the Amazon Web Services Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. You are also responsible for the security configuration and management tasks for the Amazon Web Services services that you use. For more information about data privacy, see the Data Privacy FAQ.

For data protection purposes, we recommend that you protect Amazon Web Services account credentials and set up individual users with Amazon IAM Identity Center or Amazon Identity and Access Management (IAM). That way, each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with Amazon resources. We require TLS 1.2 and recommend TLS 1.3.
- Set up API and user activity logging with Amazon CloudTrail. For information about using CloudTrail trails to capture Amazon activities, see Working with CloudTrail trails in the *Amazon CloudTrail User Guide*.
- Use Amazon encryption solutions, along with all default security controls within Amazon Web Services services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing sensitive data that is stored in Amazon S3.
- If you require FIPS 140-3 validated cryptographic modules when accessing Amazon through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see Federal Information Processing Standard (FIPS) 140-3.

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form text fields such as a **Name** field. This includes when you work with Network Synthetic Monitor or other Amazon Web Services services using the console, API, Amazon CLI, or Amazon SDKs. Any data that you enter into tags or free-form text fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

Infrastructure Security in Network Synthetic Monitor

As a managed service, Network Synthetic Monitor is protected by the Amazon global network security procedures that are described in the Amazon Web Services: Overview of Security Processes whitepaper.

You use Amazon published API calls to access Network Synthetic Monitor through the network. Clients must support Transport Layer Security (TLS) 1.0 or later. We recommend TLS 1.2 or later. Clients must also support cipher suites with perfect forward secrecy (PFS) such as DHE (Ephemeral Diffie-Hellman) or ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the [Amazon Security Token Service](#) (Amazon STS) to generate temporary security credentials to sign requests.

Identity and access management for Network Synthetic Monitor

Amazon Identity and Access Management (IAM) is an Amazon service that helps an administrator securely control access to Amazon resources. IAM administrators control who can be authenticated (signed in) and authorized (have permissions) to use Network Synthetic Monitor resources. IAM is an Amazon service that you can use with no additional charge. You can use features of IAM to allow other users, services, and applications to use your Amazon resources fully or in a limited way, without sharing your security credentials.

By default, IAM users don't have permission to create, view, or modify Amazon resources. To allow an IAM user to access resources, such as a global network, and perform tasks, you must:

- Create an IAM policy that grants the user permission to use the specific resources and API actions they need
- Attach the policy to the IAM user or to the group to which the user belongs

When you attach a policy to a user or group of users, it allows or denies the user permissions to perform the specified tasks on the specified resources.

Condition keys

The `Condition` element (or Condition block) lets you specify conditions in which a statement is in effect. The Condition element is optional. You can build conditional expressions that use condition operators, such as equals or less than, to match the condition in the policy with values in the request. For more information, see [IAM JSON policy elements: Condition operators](#) in the *Amazon Identity and Access Management User Guide*.

If you specify multiple `Condition` elements in a statement, or multiple keys in a single `Condition` element, Amazon evaluates them using a logical AND operation. If you specify multiple values for a single condition key, Amazon evaluates the condition using a logical `OR` operation. All of the conditions must be met before the statement's permissions are granted.

You can also use placeholder variables when you specify conditions. For example, you can grant an IAM user permission to access a resource only if it is tagged with their IAM user name.

You can attach tags to Network Synthetic Monitor resources or pass tags in a request to Cloud WAN. To control access based on tags, you provide tag information in the condition element of a policy using the `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys. See IAM JSON policy elements: Condition in the *Amazon Identity and Access Management User Guide* for more information.

To see all Amazon global condition keys, see Amazon global condition context keys in the *Amazon Identity and Access Management User Guide*.

Tag core network resources

A tag is a metadata label that either you or Amazon assigns to an Amazon resource. Each tag consists of a key and a value. For tags that you assign, you define the key and the value. For example, you might define the key as `purpose` and the value as `test` for one resource. Tags help you do the following:

- Identify and organize your Amazon resources. Many Amazon services support tagging, so you can assign the same tag to resources from different services to indicate that the resources are related.
- Control access to your Amazon resources. For more information, see Controlling access to Amazon resources using tags in the *Amazon Identify and Access Management User Guide*.

How Network Synthetic Monitor works with IAM

Before you use IAM to manage access to Network Synthetic Monitor, learn what IAM features are available to use with Network Synthetic Monitor.

IAM features you can use with Network Synthetic Monitor

IAM feature	Network Synthetic Monitor support
Identity-based policies	Yes

IAM feature	Network Synthetic Monitor support
Resource-based policies	No
Policy actions	Yes
Policy resources	Yes
Policy condition keys	Yes
ACLs	No
ABAC (tags in policies)	Partial
Temporary credentials	Yes
Principal permissions	Yes
Service roles	No
Service-linked roles	Yes

To get a high-level view of how Network Synthetic Monitor and other Amazon services work with most IAM features, see Amazon services that work with IAM in the *IAM User Guide*.

Identity-based policies for Network Synthetic Monitor

Supports identity-based policies: Yes

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Define custom IAM permissions with customer managed policies in the *IAM User Guide*.

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. You can't specify the principal in an identity-based policy because it applies to the user or role to which it is attached. To learn about all of the elements that you can use in a JSON policy, see IAM JSON policy elements reference in the *IAM User Guide*.

Identity-based policy examples for Network Synthetic Monitor

To view examples of Network Synthetic Monitor identity-based policies, see Identity-based policy examples for Amazon CloudWatch.

Resource-based policies within Network Synthetic Monitor

Supports resource-based policies: No

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or Amazon Web Services services.

To enable cross-account access, you can specify an entire account or IAM entities in another account as the principal in a resource-based policy. Adding a cross-account principal to a resource-based policy is only half of establishing the trust relationship. When the principal and the resource are in different Amazon Web Services accounts, an IAM administrator in the trusted account must also grant the principal entity (user or role) permission to access the resource. They grant permission by attaching an identity-based policy to the entity. However, if a resource-based policy grants access to a principal in the same account, no additional identity-based policy is required. For more information, see Cross account resource access in IAM in the *IAM User Guide*.

Policy actions for Network Synthetic Monitor

Supports policy actions: Yes

Administrators can use Amazon JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Action` element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Policy actions usually have the same name as the associated Amazon API operation. There are some exceptions, such as *permission-only actions* that don't have a matching API operation. There are also some operations that require multiple actions in a policy. These additional actions are called *dependent actions*.

Include actions in a policy to grant permissions to perform the associated operation.

To see a list of Network Synthetic Monitor actions, see Actions defined by Network Synthetic Monitor in the *Service Authorization Reference*.

Policy actions in Network Synthetic Monitor use the following prefix before the action:

```
networkmonitor
```

To specify multiple actions in a single statement, separate them with commas.

```
"Action": [
      "networkmonitor:action1",
      "networkmonitor:action2"
         ]
```

To view examples of Network Synthetic Monitor identity-based policies, see Identity-based policy examples for Amazon CloudWatch.

Policy resources for Network Synthetic Monitor

Supports policy resources: Yes

Administrators can use Amazon JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Resource` JSON policy element specifies the object or objects to which the action applies. Statements must include either a `Resource` or a `NotResource` element. As a best practice, specify a resource using its Amazon Resource Name (ARN). You can do this for actions that support a specific resource type, known as *resource-level permissions*.

For actions that don't support resource-level permissions, such as listing operations, use a wildcard (*) to indicate that the statement applies to all resources.

```
"Resource": "*"
```

To see a list of Network Synthetic Monitor resource types and their ARNs, see Resources defined by Network Synthetic Monitor in the *Service Authorization Reference*. To learn with which actions you can specify the ARN of each resource, see Actions defined by Network Synthetic Monitor.

Policy condition keys for Network Synthetic Monitor

Supports service-specific policy condition keys: Yes

Administrators can use Amazon JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Condition` element (or `Condition` *block*) lets you specify conditions in which a statement is in effect. The `Condition` element is optional. You can create conditional expressions that use condition operators, such as equals or less than, to match the condition in the policy with values in the request.

If you specify multiple `Condition` elements in a statement, or multiple keys in a single `Condition` element, Amazon evaluates them using a logical `AND` operation. If you specify multiple values for a single condition key, Amazon evaluates the condition using a logical `OR` operation. All of the conditions must be met before the statement's permissions are granted.

You can also use placeholder variables when you specify conditions. For example, you can grant an IAM user permission to access a resource only if it is tagged with their IAM user name. For more information, see IAM policy elements: variables and tags in the *IAM User Guide*.

Amazon supports global condition keys and service-specific condition keys. To see all Amazon global condition keys, see Amazon global condition context keys in the *IAM User Guide*.

To see a list of Network Synthetic Monitor condition keys, see Condition keys for Network Synthetic Monitor in the *Service Authorization Reference*. To learn with which actions and resources you can use a condition key, see Actions defined by Network Synthetic Monitor.

ACLs in Network Synthetic Monitor

Supports ACLs: No

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

ABAC with Network Synthetic Monitor

Supports ABAC (tags in policies): Partial

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes. In Amazon, these attributes are called *tags*. You can attach tags to IAM entities (users or roles) and to many Amazon resources. Tagging entities and resources is the first step of ABAC. Then you design ABAC policies to allow operations when the principal's tag matches the tag on the resource that they are trying to access.

ABAC is helpful in environments that are growing rapidly and helps with situations where policy management becomes cumbersome.

To control access based on tags, you provide tag information in the [condition element](#) of a policy using the `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys.

If a service supports all three condition keys for every resource type, then the value is **Yes** for the service. If a service supports all three condition keys for only some resource types, then the value is **Partial**.

For more information about ABAC, see Define permissions with ABAC authorization in the *IAM User Guide*. To view a tutorial with steps for setting up ABAC, see Use attribute-based access control (ABAC) in the *IAM User Guide*.

Using temporary credentials with Network Synthetic Monitor

Supports temporary credentials: Yes

Some Amazon Web Services services don't work when you sign in using temporary credentials. For additional information, including which Amazon Web Services services work with temporary credentials, see Amazon Web Services services that work with IAM in the *IAM User Guide*.

You are using temporary credentials if you sign in to the Amazon Web Services Management Console using any method except a user name and password. For example, when you access Amazon using your company's single sign-on (SSO) link, that process automatically creates temporary credentials. You also automatically create temporary credentials when you sign in to the console as a user and then switch roles. For more information about switching roles, see Switch from a user to an IAM role (console) in the *IAM User Guide*.

You can manually create temporary credentials using the Amazon CLI or Amazon API. You can then use those temporary credentials to access Amazon. Amazon recommends that you dynamically generate temporary credentials instead of using long-term access keys. For more information, see Temporary security credentials in IAM.

Cross-service principal permissions for Network Synthetic Monitor

Supports forward access sessions (FAS): Yes

When you use an IAM user or role to perform actions in Amazon, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an Amazon Web Services service, combined with the requesting Amazon Web Services service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other Amazon Web Services services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see Forward access sessions.

Service roles for Network Synthetic Monitor

Supports service roles: No

A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Create a role to delegate permissions to an Amazon Web Services service in the *IAM User Guide*.

> **Warning**
>
> Changing the permissions for a service role might break Network Synthetic Monitor functionality. Edit service roles only when Network Synthetic Monitor provides guidance to do so.

Using a service-linked role for Network Synthetic Monitor

Supports service-linked roles: Yes

A service-linked role is a type of service role that is linked to an Amazon Web Services service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your Amazon Web Services account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

For details about creating or managing service-linked roles, see Amazon services that work with IAM. Find a service in the table that includes a `Yes` in the **Service-linked role** column. Choose the **Yes** link to view the service-linked role documentation for that service.

Identity-based policy examples for Network Synthetic Monitor

By default, users and roles don't have permission to create or modify Network Synthetic Monitor resources. They also can't perform tasks by using the Amazon Web Services Management Console, Amazon Command Line Interface (Amazon CLI), or Amazon API. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

To learn how to create an IAM identity-based policy by using these example JSON policy documents, see Create IAM policies (console) in the *IAM User Guide*.

For details about actions and resource types defined by Network Synthetic Monitor, including the format of the ARNs for each of the resource types, see Actions, resources, and condition keys for Network Synthetic Monitor in the *Service Authorization Reference*.

Topics

- Policy best practices
- Using the Network Synthetic Monitor console
- Allow users to view their own permissions
- Troubleshooting Network Synthetic Monitor identity and access

Policy best practices

Identity-based policies determine whether someone can create, access, or delete Network Synthetic Monitor resources in your account. These actions can incur costs for your Amazon Web Services account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get started with Amazon managed policies and move toward least-privilege permissions** – To get started granting permissions to your users and workloads, use the *Amazon managed policies* that grant permissions for many common use cases. They are available in your Amazon Web Services account. We recommend that you reduce permissions further by defining Amazon customer managed policies that are specific to your use cases. For more information, see Amazon managed policies or Amazon managed policies for job functions in the *IAM User Guide*.
- **Apply least-privilege permissions** – When you set permissions with IAM policies, grant only the permissions required to perform a task. You do this by defining the actions that can be taken on specific resources under specific conditions, also known as *least-privilege permissions*. For more information about using IAM to apply permissions, see Policies and permissions in IAM in the *IAM User Guide*.
- **Use conditions in IAM policies to further restrict access** – You can add a condition to your policies to limit access to actions and resources. For example, you can write a policy condition to specify that all requests must be sent using SSL. You can also use conditions to grant access to service actions if they are used through a specific Amazon Web Services service, such as Amazon CloudFormation. For more information, see IAM JSON policy elements: Condition in the *IAM User Guide*.

- **Use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions** – IAM Access Analyzer validates new and existing policies so that the policies adhere to the IAM policy language (JSON) and IAM best practices. IAM Access Analyzer provides more than 100 policy checks and actionable recommendations to help you author secure and functional policies. For more information, see Validate policies with IAM Access Analyzer in the *IAM User Guide*.
- **Require multi-factor authentication (MFA)** – If you have a scenario that requires IAM users or a root user in your Amazon Web Services account, turn on MFA for additional security. To require MFA when API operations are called, add MFA conditions to your policies. For more information, see Secure API access with MFA in the *IAM User Guide*.

For more information about best practices in IAM, see Security best practices in IAM in the *IAM User Guide*.

Using the Network Synthetic Monitor console

To access the Network Synthetic Monitor console, you must have a minimum set of permissions. These permissions must allow you to list and view details about the Network Synthetic Monitor resources in your Amazon Web Services account. If you create an identity-based policy that is more restrictive than the minimum required permissions, the console won't function as intended for entities (users or roles) with that policy.

You don't need to allow minimum console permissions for users that are making calls only to the Amazon CLI or the Amazon API. Instead, allow access to only the actions that match the API operation that they're trying to perform.

To ensure that users and roles can still use the Network Synthetic Monitor console, also attach the Network Synthetic Monitor `ConsoleAccess` or `ReadOnly` Amazon managed policy to the entities. For more information, see Adding permissions to a user in the *IAM User Guide*.

Allow users to view their own permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the Amazon CLI or Amazon API.

```
{
    "Version": "2012-10-17",
```

```
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws-cn:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

Troubleshooting Network Synthetic Monitor identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with Network Synthetic Monitor and IAM.

Topics

- I am not authorized to perform an action in Network Synthetic Monitor
- I am not authorized to perform iam:PassRole
- I want to allow people outside of my Amazon Web Services account to access my Network Synthetic Monitor resources

I am not authorized to perform an action in Network Synthetic Monitor

If you receive an error that you're not authorized to perform an action, your policies must be updated to allow you to perform the action.

The following example error occurs when the `mateojackson` IAM user tries to use the console to view details about a fictional *my-example-widget* resource but doesn't have the fictional `networkmonitor:GetWidget` permissions.

```
User: arn:aws-cn:iam::123456789012:user/mateojackson is not authorized to perform:
 networkmonitor:GetWidget on resource: my-example-widget
```

In this case, the policy for the `mateojackson` user must be updated to allow access to the *my-example-widget* resource by using the `networkmonitor:GetWidget` action.

If you need help, contact your Amazon administrator. Your administrator is the person who provided you with your sign-in credentials.

I am not authorized to perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, your policies must be updated to allow you to pass a role to Network Synthetic Monitor.

Some Amazon Web Services services allow you to pass an existing role to that service instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named `marymajor` tries to use the console to perform an action in Network Synthetic Monitor. However, the action requires the service to have permissions that are granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws-cn:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

In this case, Mary's policies must be updated to allow her to perform the `iam:PassRole` action.

If you need help, contact your Amazon administrator. Your administrator is the person who provided you with your sign-in credentials.

I want to allow people outside of my Amazon Web Services account to access my Network Synthetic Monitor resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether Network Synthetic Monitor supports these features, see How Amazon CloudWatch works with IAM.
- To learn how to provide access to your resources across Amazon Web Services accounts that you own, see Providing access to an IAM user in another Amazon Web Services account that you own in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party Amazon Web Services accounts, see Providing access to Amazon Web Services accounts owned by third parties in the *IAM User Guide*.
- To learn how to provide access through identity federation, see Providing access to externally authenticated users (identity federation) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see Cross account resource access in IAM in the *IAM User Guide*.

Amazon managed policies for Network Synthetic Monitor

To add permissions to users, groups, and roles, it is easier to use Amazon managed policies than to write policies yourself. It takes time and expertise to create IAM customer managed policies that provide your team with only the permissions they need. To get started quickly, you can use our Amazon managed policies. These policies cover common use cases and are available in your Amazon account. For more information about Amazon managed policies, see Amazon managed policies in the *IAM User Guide*.

Amazon services maintain and update Amazon managed policies. You can't change the permissions in Amazon managed policies. Services occasionally add additional permissions to an Amazon managed policy to support new features. This type of update affects all identities (users, groups, and roles) where the policy is attached. Services are most likely to update an Amazon managed policy when a new feature is launched or when new operations become available. Services do not remove permissions from an Amazon managed policy, so policy updates won't break your existing permissions.

Additionally, Amazon supports managed policies for job functions that span multiple services. For example, the `ReadOnlyAccess` Amazon managed policy provides read-only access to all Amazon services and resources. When a service launches a new feature, Amazon adds read-only permissions for new operations and resources. For a list and descriptions of job function policies, see Amazon managed policies for job functions in the *IAM User Guide*.

Amazon managed policy: CloudWatchNetworkMonitorServiceRolePolicy

The `CloudWatchNetworkMonitorServiceRolePolicy` is attached to a service-linked role that allows the service to perform actions on your behalf and access resources associated with CloudWatch Network Synthetic Monitor. You cannot attach this policy to your IAM identities. For more information, see Using a service-linked role for Network Synthetic Monitor.

Network Synthetic Monitor updates to Amazon managed policies

View details about updates to Amazon managed policies for Network Synthetic Monitor since this service began tracking these changes in November 2023.

Change	Description	Date
CloudWatchNetworkMonitorServiceRolePolicy: New policy.	New policy added to Network Synthetic Monitor.	November 27, 2023
the section called "`AWSServiceRoleForNetworkMonitor`". New role.	New role added to Network Synthetic Monitor.	November 27, 2023

IAM permissions for Network Synthetic Monitor

To use Network Synthetic Monitor users must have the correct permissions.

For more information about security in Amazon CloudWatch, see Identity and access management for Amazon CloudWatch.

Permissions required to view a monitor

To view a monitor for Network Synthetic Monitor in the Amazon Web Services Management Console, you must be signed in as a user or role that has the following permissions:

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:GetMetricData",
                "networkmonitor:Get*",
                "networkmonitor:List*"
                ],
            "Resource": "*"
        }
    ]
}
```

Permissions required to create a monitor

To create a monitor in Network Synthetic Monitor, users must have permission to create a service-linked role that is associated with Network Synthetic Monitor. To learn more about the service-linked role, see [Using a service-linked role for Network Synthetic Monitor](#).

To create a monitor for Network Synthetic Monitor in the Amazon Web Services Management Console, you must be signed in as a user or role that has the permissions included in the following policy.

> **Note**
>
> If you create an identity-based permissions policy that is more restrictive, users with that policy won't be able to create a monitor.

```
{
"Version": "2012-10-17",
    "Statement": [
        {
```

```
            "Effect": "Allow",
            "Action": [
                "networkmonitor:*"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::*:role/aws-service-role/
networkmonitor.amazonaws.com/AWSServiceRoleForNetworkMonitor",
            "Condition": {
                "StringLike": {
                    "iam:AWSServiceName": "networkmonitor.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:AttachRolePolicy",
                "iam:GetRole",
                "iam:PutRolePolicy"
            ],
            "Resource": "arn:aws:iam::*:role/aws-service-role/
networkmonitor.amazonaws.com/AWSServiceRoleForNetworkMonitor"
        },
        {
            "Action": [
                "ec2:CreateSecurityGroup",
                "ec2:CreateNetworkInterface",
                "ec2:CreateTags"
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
    ]
}
```

Using a service-linked role for Network Synthetic Monitor

Network Synthetic Monitor uses the following service-linked role for the permissions that it requires to call other Amazon services on your behalf:

- [AWSServiceRoleForNetworkMonitor]

AWSServiceRoleForNetworkMonitor

Network Synthetic Monitor uses the service-linked role named `AWSServiceRoleForNetworkMonitor` to update and manage monitors.

The `AWSServiceRoleForNetworkMonitor` service-linked role trusts the following service to assume the role:

- `networkmonitor.amazonaws.com`

The `CloudWatchNetworkMonitorServiceRolePolicy` is attached to the service linked role and grants access for the service to access VPC and EC2 resources in your account, as well as manage the monitors that you create.

Permissions groupings

The policy is grouped into the following sets of permissions:

- **`cloudwatch`** - This allows the service principal to publish network monitoring metrics to CloudWatch resources.
- **`ec2`** - This allows the service principal to describe VPCs and subnets in your account to create or update monitors and probes. This also allows the service principal to create, modify, and delete security groups, network interfaces, and their associated permissions to configure the monitor or probe to send monitoring traffic to your endpoints.

To view the permissions for this policy, see [CloudWatchNetworkMonitorServiceRolePolicy] in the *Amazon Managed Policy Reference*.

Create the service-linked role

`AWSServiceRoleForNetworkMonitor`

You don't need to manually create the `AWSServiceRoleForNetworkMonitor` role.

- Network Synthetic Monitor creates the `AWSServiceRoleForNetworkMonitor` role when you create your first monitor with the feature. This role then applies to all additional monitors that you create.

To create a service-linked role on your behalf, you must have the required permissions. For more information, see [Service-Linked Role Permissions](#) in the *IAM User Guide*.

Edit the service-linked role

You can edit the `AWSServiceRoleForNetworkMonitor` descriptions using IAM. For more information, see [Editing a Service-Linked Role](#) in the *IAM User Guide*.

Delete the service-linked role

If you no longer need to use Network Synthetic Monitor, we recommend that you delete the `AWSServiceRoleForNetworkMonitor` role.

You can delete these service-linked roles only after you delete your monitors. For more information, see [Delete a monitor](#).

You can use the IAM console, the IAM CLI, or the IAM API to delete service-linked roles. For more information, see [Deleting a Service-Linked Role](#) in the *IAM User Guide*.

After you delete `AWSServiceRoleForNetworkMonitor` Network Synthetic Monitor will create the role again when you create a new monitor.

Supported Regions for the Network Synthetic Monitor service-linked role

Network Synthetic Monitor supports the service-linked role in all of Amazon Web Services Regions where the service is available. For more information, see [Amazon endpoints](#) in the *Amazon Web Services General Reference*.

Delete the service-linked role

If you no longer need to use Network Synthetic Monitor, we recommend that you delete the `AWSServiceRoleForNetworkMonitor` role.

You can delete these service-linked roles only after you delete your monitors. For more information, see [Delete a monitor](#).

You can use the IAM console, the IAM CLI, or the IAM API to delete service-linked roles. For more information, see [Deleting a Service-Linked Role](#) in the *IAM User Guide*.

After you delete `AWSServiceRoleForNetworkMonitor` Network Synthetic Monitor will create the role again when you create a new monitor.

Network Synthetic Monitor quotas

The following are the Network Synthetic Monitor quotas:

Quota	Default	Adjustable
Maximum number of monitors per account per Amazon Web Services Region	100	Yes
Maximum number of probes per monitor	24	Yes
Maximum number of probes per subnet per monitor	4	Yes

CloudWatch observability solutions

CloudWatch observability solutions offer a catalog of readily available configurations to help you quickly implement monitoring for various Amazon services and common workloads, such as Java Virtual Machines (JVM), Apache Kafka, Apache Tomcat, and NGINX. These solutions provide focused guidance on key monitoring tasks, including the installation and configuration of the CloudWatch agent, deployment of pre-defined custom dashboards, and setup of metric alarms. They are designed to assist developers and operations teams in leveraging Amazon monitoring and observability tools more effectively.

The solutions include guidance on when to use specific observability features like Detailed Monitoring metrics for infrastructure, Container Insights for container monitoring, and Application Signals for application monitoring. By providing working examples and practical configurations, these solutions aim to simplify the initial setup process, allowing you to establish functional monitoring more quickly and customize as needed for their specific requirements.

To get started with observability solutions, visit the observability solutions page in the CloudWatch console.

For open-source solutions that work with Amazon Managed Grafana, see Amazon Managed Grafana solutions

Solutions that require CloudWatch agent are detailed below:

Topics

- CloudWatch solution: JVM workload on Amazon EC2
- CloudWatch solution: NGINX workload on Amazon EC2
- CloudWatch solution: NVIDIA GPU workload on Amazon EC2
- CloudWatch solution: Kafka workload on Amazon EC2
- CloudWatch solution: Tomcat workload on Amazon EC2
- CloudWatch solution: Amazon EC2 health

How do solution dashboards work?

The dashboards for CloudWatch solutions use search-powered variables (dropdowns) that allow you to explore and visualize different aspects of your workloads dynamically.

By combining the flexibility of search-powered variables with the pre-configured metric widgets, the dashboard provides deep insights into your workloads, enabling proactive monitoring, troubleshooting, and optimization. This dynamic approach ensures that you can quickly adapt the dashboard to your specific monitoring needs, without the need for extensive customization or configuration.

Do solutions support cross-Region observability?

CloudWatch solution dashboards display metrics of the Region where the solution dashboard is created. However, the solution dashboard doesn't display metrics across multiple Regions. If you have a use case to view data from multiple Regions in a single dashboard, you'll need to customize the dashboard JSON to add the Regions that you want to view. To do this, use the `region` attribute of the metric format to query the metrics from different Regions. For more information about modifying dashboard JSON, see Metric Widget: Format for Each Metric in the Array.

Do solution dashboards support Cross-account cross-Region CloudWatch console?

When using CloudWatch cross-account observability, solution dashboards in the central monitoring account display metrics from source accounts in the same Region. To differentiate metrics for similar workloads across accounts, provide unique grouping dimension values in agent configurations. For instance, assign distinct `ClusterName` values for Kafka brokers in different accounts for Kafka workload, enabling precise cluster selection and metric viewing in the dashboard.

Do solution dashboards support CloudWatch cross-account observability?

If you have enabled cross-account using Cross-account cross-Region CloudWatch console, you won't be able to use the solution dashboard created in the monitoring account to view metrics from source accounts. Instead, you'll need to create dashboards in the respective source accounts. However, you can create the dashboard in the source account and view it from the monitoring account by switching the account ID setting in the console.

What are the limitations for a solution dashboard?

Solution dashboards leverage Search expressions to filter and analyze metrics for the workloads. This enables dynamic views based on dropdown option selections. These search expressions might return more than 500 time series, but each dashboard widget can't display more than 500 time series. If a metric search in the solution dashboard results in more than 500 time series across all Amazon EC2; instances, the graph displaying the top contributors might show inaccurate results. For more information about search expressions, see CloudWatch search expression syntax.

CloudWatch displays the metric information on the dashboards if you click the `i` icon on the dashboard widget. However, this currently doesn't work for dashboard widgets that use search expressions. The solution dashboards use search expressions, so you won't be able to see the metric description in the dashboard.

Can I customize the agent configuration or the dashboard provided by a solution?

You can customize the agent configuration and the dashboard. Be aware that if you customize the agent configuration, you must update the dashboard accordingly or it will display empty metric widgets. Also be aware that if CloudWatch releases a new version of a solution, you might have to repeat your customizations if you apply the newer version of the solution.

How are solutions versioned?

Each solution provides the most up-to-date instructions and resources. We always recommend using the latest version available. While the solutions themselves are not versioned, the associated artifacts (such as Amazon CloudFormation templates for dashboards and agent installations) are versioned.

You can identify the version of a previously deployed artifact by checking the Amazon CloudFormation template's description field or the filename of the template you downloaded. To determine if you're using the latest version, compare your deployed version with the one currently referenced in the solution documentation.

CloudWatch solution: JVM workload on Amazon EC2

This solution helps you configure out-of-the-box metric collection using CloudWatch agents for JVM application running on EC2 instances. Additionally, it helps you set up a pre-configured CloudWatch dashboard. For general information about all CloudWatch observability solutions, see CloudWatch observability solutions.

Topics

Requirements

This solution is relevant for the following conditions:

- Supported versions: Java LTS versions 8, 11, 17, and 21
- Compute: Amazon EC2
- Supports up to 500 EC2 instances across all JVM workloads in a given Amazon Web Services Region
- Latest version of CloudWatch agent
- SSM agent installed on EC2 instance

> **Note**
>
> Amazon Systems Manager (SSM agent) is pre-installed on some Amazon Machine Images (AMIs) provided by Amazon and trusted third-parties. If the agent isn't installed, you can install it manually using the procedure for your operating system type.
>
> - Manually installing and uninstalling SSM Agent on EC2 instances for Linux
> - Manually installing and uninstalling SSM Agent on EC2 instances for macOS
> - Manually installing and uninstalling SSM Agent on EC2 instances for Windows Server

Benefits

The solution delivers JVM monitoring, providing valuable insights for the following use cases:

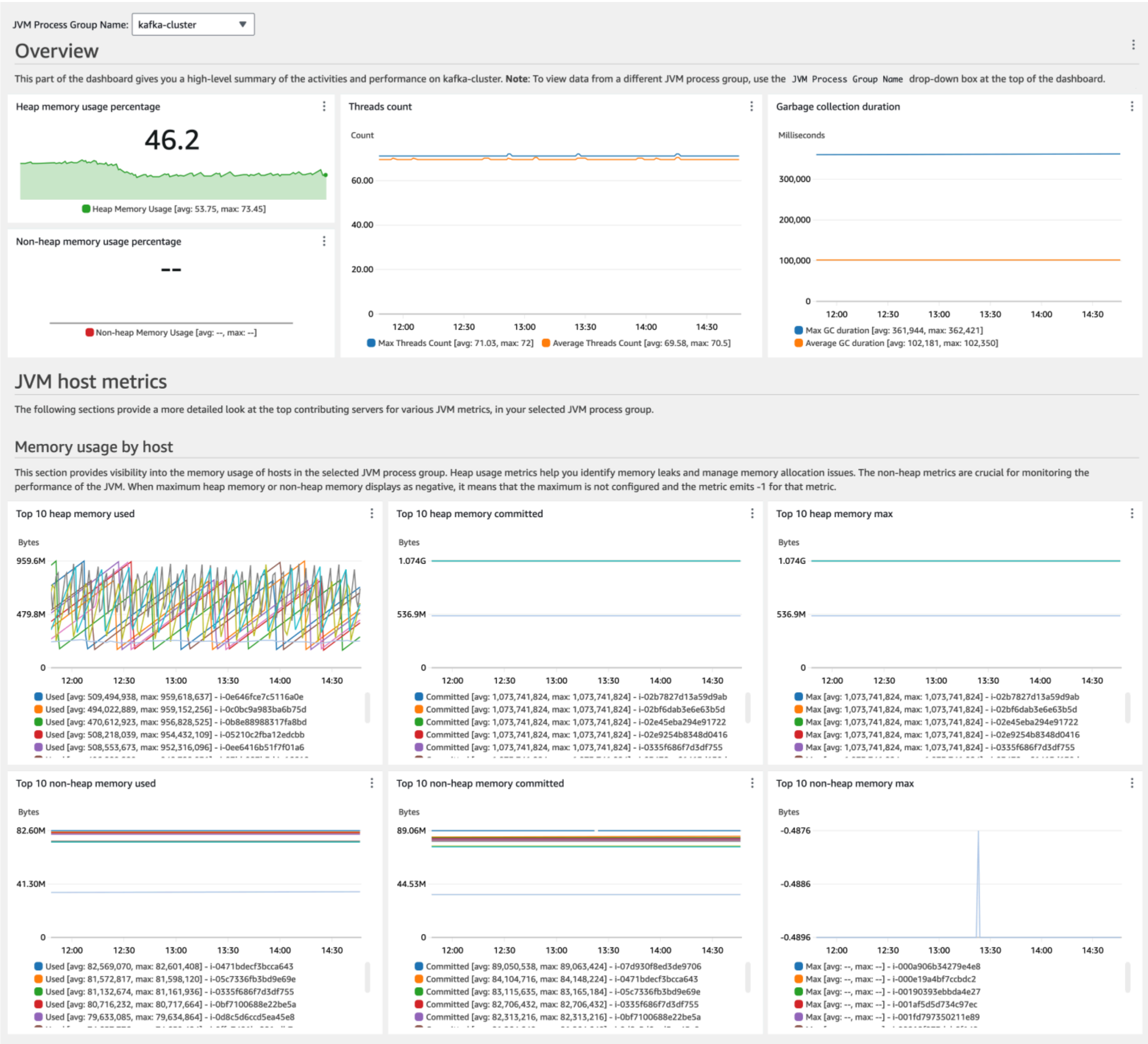
- Monitor JVM heap and non-heap memory usage.
- Analyze thread and class loading for concurrency issues.
- Track garbage collection to identify memory leaks.
- Switch between different JVM applications configured via the solution under the same account.

Below are the key advantages of the solution:

- Automates metric collection for JVM using CloudWatch agent configuration, eliminating manual instrumentation.
- Provides a pre-configured, consolidated CloudWatch dashboard for JVM metrics. The dashboard will automatically handle metrics from new JVM EC2 instances configured using the solution,

even if those metrics don't exist when you first create the dashboard. It also allows you to group the metrics into logical applications for easier focus and management.

The following image is an example of the dashboard for this solution.



Costs

This solution creates and uses resources in your account. You are charged for standard usage, including the following:

- All metrics collected by the CloudWatch agent are charged as custom metrics. The number of metrics used by this solution depends on the number of EC2 hosts.
  - Each JVM host configured for the solution publishes a total of 18 metrics plus one metric (`disk_used_percent`) for which the metric count depends on the number of paths for the host.
- One custom dashboard.
- API operations requested by the CloudWatch agent to publish the metrics. With the default configuration for this solution, the CloudWatch agent calls the **PutMetricData** once every minute for each EC2 host. This means the **PutMetricData** API will be called `30*24*60=43,200` in a 30-day month for each EC2 host.

For more information about CloudWatch pricing, see [Amazon CloudWatch Pricing](https://aws.amazon.com/cloudwatch/pricing/).

The pricing calculator can help you estimate approximate monthly costs for using this solution.

To use the pricing calculator to estimate your monthly solution costs

1. Open the [Amazon CloudWatch pricing calculator](https://calculator.aws/#/createCalculator/CloudWatch).
2. For **Choose a Region**, select the Region where you would like to deploy the solution.
3. In the **Metrics** section, for **Number of metrics**, enter **`(18 + average number of disk paths per EC2 host) * number of EC2 instances configured for this solution`**.
4. In the **APIs** section, for **Number of API requests**, enter **`43200 * number of EC2 instances configured for this solution`**.

   By default, the CloudWatch agent performs one **PutMetricData** operation each minute for each EC2 host.
5. In the **Dashboards and Alarms** section, for **Number of Dashboards**, enter **1**.
6. You can see your monthly estimated costs at the bottom of the pricing calculator.

CloudWatch agent configuration for this solution

The CloudWatch agent is software that runs continuously and autonomously on your servers and in containerized environments. It collects metrics, logs, and traces from your infrastructure and applications and sends them to CloudWatch and X-Ray.

For more information about the CloudWatch agent, see [Collect metrics, logs, and traces with the CloudWatch agent](#).

The agent configuration in this solution collects the foundational metrics for the solution. The CloudWatch agent can be configured to collect more JVM metrics than the dashboard displays by default. For a list of all JVM metrics that you can collect, see [Collect JVM metrics](#). For general information about CloudWatch agent configuration, see [Metrics collected by the CloudWatch agent](#).

Expose JMX ports for the JVM application

The CloudWatch agent relies on JMX to collect the metrics related to the JVM process. To make this possible, you must expose the JMX port from your JVM application. Instructions for exposing the JMX port depend on the workload type you are using for your JVM application. See the documentation for your application to find these instructions.

In general, to enable a JMX port for monitoring and management, you would set the following system properties for your JVM application. Be sure to specify an unused port number. The following example sets up unauthenticated JMX. If your security policies/requirements require you to enable JMX with password authentication or SSL for remote access, refer to the [JMX documentation](#) to set the required property.

```
-Dcom.sun.management.jmxremote
-Dcom.sun.management.jmxremote.port=port-number
-Dcom.sun.management.jmxremote.authenticate=false
-Dcom.sun.management.jmxremote.ssl=false
```

Review the starting scripts and configuration files of your application to find the best place to add these arguments. When you run a `.jar` file from the command line, this command could look like the following, where *`pet-search.jar`* is the name of the application jar.

```
$ java -jar -Dcom.sun.management.jmxremote -Dcom.sun.management.jmxremote.port=9999 -
Dcom.sun.management.jmxremote.authenticate=false -
Dcom.sun.management.jmxremote.ssl=false pet-search.jar
```

Agent configuration for this solution

The metrics collected by the agent are defined in the agent configuration. The solution provides agent configurations to collect the recommended metrics with suitable dimensions for the solution’s dashboard.

The steps for deploying the solution are described later in Deploy the agent for your solution. The following information is intended to help you understand how to customize the agent configuration for your environment.

You must customize some parts of the following agent configuration for your environment:

- The JMX port number is the port number that you configured in the previous section of this documentation. It is in the `endpoint` line in the configuration.
- `ProcessGroupName`– Provide meaningful names for the `ProcessGroupName` dimension. These names should represent the cluster, application, or services grouping for EC2 instances running the same application or process. This helps you to group metrics from instances belonging to the same JVM process group, providing a unified view of cluster, application, and service performance in the solution dashboard.

For example, if you have two Java applications running in the same account, one for the `order-processing` application and another for the `inventory-management` application, you should set the `ProcessGroupName` dimensions accordingly in the agent configuration of each instance.

- For the `order-processing` application instances, set `ProcessGroupName=order-processing`.
- For the `inventory-management` application instances, set `ProcessGroupName=inventory-management`.

When you follow these guidelines, the solution dashboard will automatically group the metrics based on the `ProcessGroupName` dimension. The dashboard will include dropdown options to select and view metrics for a specific process group, allowing you to monitor the performance of individual process groups separately.

Agent configuration for JVM hosts

Use the following CloudWatch agent configuration on EC2 instances where your Java applications is deployed. Configuration will be stored as a parameter in SSM's Parameter Store, as detailed later in Step 2: Store the recommended CloudWatch agent configuration file in Systems Manager Parameter Store.

Replace *`ProcessGroupName`* with the name of your process group. Replace *`port-number`* with the JMX port of your Java application. If JMX was enabled with password authentication or SSL for remote access, see [Collect Java Management Extensions (JMX) metrics] for information about setting up TLS or authorization in agent configuration as required.

The EC2 metrics shown in this configuration (configuration shown outside the JMX block) only work for Linux and macOS instances. If you are using Windows instances, you can choose to omit these metrics in the configuration. For information about metrics collected on Windows instances, see [Metrics collected by the CloudWatch agent on Windows Server instances].

```
{
  "metrics": {
    "namespace": "CWAgent",
    "append_dimensions": {
      "InstanceId": "${aws:InstanceId}"
    },
    "metrics_collected": {
      "jmx": [
        {
          "endpoint": "localhost:port-number",
          "jvm": {
            "measurement": [
              "jvm.classes.loaded",
              "jvm.gc.collections.count",
              "jvm.gc.collections.elapsed",
              "jvm.memory.heap.committed",
              "jvm.memory.heap.max",
              "jvm.memory.heap.used",
              "jvm.memory.nonheap.committed",
              "jvm.memory.nonheap.max",
              "jvm.memory.nonheap.used",
              "jvm.threads.count"
            ]
          },
          "append_dimensions": {
            "ProcessGroupName": "ProcessGroupName"
          }
        }
      ],
      "disk": {
        "measurement": [
          "used_percent"
        ]
      },
      "mem": {
```

```
          "measurement": [
            "used_percent"
          ]
        },
        "swap": {
          "measurement": [
            "used_percent"
          ]
        },
        "netstat": {
          "measurement": [
            "tcp_established",
            "tcp_time_wait"
          ]
        }
      }
    }
}
```

Deploy the agent for your solution

There are several approaches for installing the CloudWatch agent, depending on the use case. We recommend using Systems Manager for this solution. It provides a console experience and makes it simpler to manage a fleet of managed servers within a single Amazon account. The instructions in this section use Systems Manager and are intended for when you don't have the CloudWatch agent running with existing configurations. You can check whether the CloudWatch agent is running by following the steps in Verify that the CloudWatch agent is running.

If you are already running the CloudWatch agent on the EC2 hosts where the workload is deployed and managing the agent configurations, you can skip the instructions in this section and follow your existing deployment mechanism to update the configuration. Be sure to merge the agent configuration of JVM with your existing agent configuration, and then deploy the merged configuration. If you are using Systems Manager to store and manage the configuration for the CloudWatch agent, you can merge the configuration to the existing parameter value. For more information, see Managing CloudWatch agent configuration files.

> **Note**
>
> Using Systems Manager to deploy the following CloudWatch agent configurations will replace or overwrite any existing CloudWatch agent configuration on your EC2 instances.

You can modify this configuration to suit your unique environment or use case. The metrics defined in this solution are the minimum required for the recommended dashboard.

The deployment process includes the following steps:

- Step 1: Ensure that the target EC2 instances have the required IAM permissions.
- Step 2: Store the recommended agent configuration file in the Systems Manager Parameter Store.
- Step 3: Install the CloudWatch agent on one or more EC2 instances using an Amazon CloudFormation stack.
- Step 4: Verify the agent setup is configured properly.

Step 1: Ensure the target EC2 instances have the required IAM permissions

You must grant permission for Systems Manager to install and configure the CloudWatch agent. You must also grant permission for the CloudWatch agent to publish telemetry from your EC2 instance to CloudWatch. Make sure that the IAM role attached to the instance has the **CloudWatchAgentServerPolicy** and **AmazonSSMManagedInstanceCore** IAM policies attached.

- To create a role, see Create IAM roles to use with the CloudWatch agent on Amazon EC2 instances.
- After the role is created, attach the role to your EC2 instances. Follow the steps in Launch an instance with an IAM role to attach a role while launching a new EC2 instance. To attach a role to an existing EC2 instance, follow the steps in Attach an IAM role to an instance.

Step 2: Store the recommended CloudWatch agent configuration file in Systems Manager Parameter Store

Parameter Store simplifies the installation of the CloudWatch agent on an EC2 instance by securely storing and managing configuration parameters, eliminating the need for hard-coded values. This ensures a more secure and flexible deployment process, enabling centralized management and easier updates to configurations across multiple instances.

Use the following steps to store the recommended CloudWatch agent configuration file as a parameter in Parameter Store.

To create the CloudWatch agent configuration file as a parameter

1. Open the Amazon Systems Manager console at https://console.amazonaws.cn/systems-manager/.
2. From the navigation pane, choose **Application Management**, **Parameter Store**.
3. Follow these steps to create a new parameter for the configuration.

    a. Choose **Create parameter**.

    b. In the **Name** box, enter a name that you'll use to reference the CloudWatch agent configuration file in later steps. For example, **`AmazonCloudWatch-JVM-Configuration`**.

    c. (Optional) In the **Description** box, type a description for the parameter.

    d. For **Parameter tier**, choose **Standard**.

    e. For **Type**, choose **String**.

    f. For **Data type**, choose **text**.

    g. In the **Value** box, paste the corresponding JSON block that was listed in Agent configuration for JVM hosts. Be sure to customize the grouping dimension value and port number as described.

    h. Choose **Create parameter**.

Step 3: Install the CloudWatch agent and apply the configuration using an Amazon CloudFormation template

You can use Amazon CloudFormation to install the agent and configure it to use the CloudWatch agent configuration that you created in the previous steps.

To install and configure the CloudWatch agent for this solution

1. Open the Amazon CloudFormation **Quick create stack** wizard using this link: https://console.aws.amazon.com/cloudformation/home?#/stacks/quickcreate?templateURL=https://aws-observability-solutions-prod-us-east-1.s3.us-east-1.amazonaws.com/CloudWatchAgent/CFN/v1.0.0/cw-agent-installation-template-1.0.0.json .
2. Verify that the selected Region on the console is the Region where the JVM workload is running.
3. For **Stack name**, enter a name to identity this stack, such as **`CWAgentInstallationStack`**.

4. In the **Parameters** section, specify the following:

   a. For **CloudWatchAgentConfigSSM**, enter the name of the Systems Manager parameter for the agent configuration that you created earlier, such as **`AmazonCloudWatch-JVM-Configuration`**.

   b. To select the target instances, you have two options.

      i. For **InstanceIds**, specify a comma-delimited list of instance IDs list of instance IDs where you want to install the CloudWatch agent with this configuration. You can list a single instance or several instances.

      ii. If you are deploying at scale, you can specify the **TagKey** and the corresponding **TagValue** to target all EC2 instances with this tag and value. If you specify a **TagKey**, you must specify a corresponding **TagValue**. (For an Auto Scaling group, specify **`aws:autoscaling:groupName`** for the **TagKey** and specify the Auto Scaling group name for the **TagValue** to deploy to all instances within the Auto Scaling group.)

      If you specify both the **InstanceIds** and the **TagKeys** parameters, the **InstanceIds** will take precedence and the tags will be ignored.

5. Review the settings, then choose **Create stack**.

If you want to edit the template file first to customize it, choose the **Upload a template file** option under **Create Stack Wizard** to upload the edited template. For more information, see [Creating a stack on Amazon CloudFormation console](). You can use the following link to download the template: [https://aws-observability-solutions-prod-us-east-1.s3.us-east-1.amazonaws.com/CloudWatchAgent/CFN/v1.0.0/cw-agent-installation-template-1.0.0.json]().

> **Note**
>
> After this step is completed, this Systems Manager parameter will be associated with the CloudWatch agents running in the targeted instances. This means that:
>
> 1. If the Systems Manager parameter is deleted, the agent will stop.
> 2. If the Systems Manager parameter is edited, the configuration changes will automatically apply to the agent at the scheduled frequency which is 30 days by default.

3. If you want to immediately apply changes to this Systems Manager parameter, you must run this step again. For more information about associations, see Working with associations in Systems Manager.

Step 4: Verify the agent setup is configured properly

You can verify whether the CloudWatch agent is installed by following the steps in Verify that the CloudWatch agent is running. If the CloudWatch agent is not installed and running, make sure you have set up everything correctly.

- Be sure you have attached a role with correct permissions for the EC2 instance as described in Step 1: Ensure the target EC2 instances have the required IAM permissions.
- Be sure you have correctly configured the JSON for the Systems Manager parameter. Follow the steps in ???.

If everything is set up correctly, then you should see the JVM metrics being published to CloudWatch. You can check the CloudWatch console to verify they are being published.

To verify that JVM metrics are being published to CloudWatch

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. Choose **Metrics**, **All metrics**.
3. Make sure you've selected the Region where you deployed the solution, and choose **Custom namespaces**, **CWAgent**.
4. Search for the metrics mentioned in Agent configuration for JVM hosts, such as `jvm.memory.heap.used`. If you see results for these metrics, then the metrics are being published to CloudWatch.

Create the JVM solution dashboard

The dashboard provided by this solution presents metrics for the underlying Java Virtual Machine (JVM) for the server. It offers an overview of the JVM by aggregating and presenting metrics across all instances, providing a high-level summary of the overall health and operational state. Additionally, the dashboard shows a breakdown of the top contributors (top 10 per metric widget) for each metric. This helps you to quickly identify outliers or instances that significantly contribute to the observed metrics.

The solution dashboard doesn't display EC2 metrics. To view EC2 metrics, you'll need to use the EC2 automatic dashboard to see EC2 vended metrics and use the EC2 console dashboard to see EC2 metrics that are collected by the CloudWatch agent. For more information about automatic dashboards for Amazon services, see Viewing a CloudWatch dashboard for a single Amazon service.

To create the dashboard, you can use the following options:

- Use CloudWatch console to create the dashboard.
- Use Amazon CloudFormation console to deploy the dashboard.
- Download the Amazon CloudFormation infrastructure as code and integrate it as part of your continuous integration (CI) automation.

By using the CloudWatch console to create a dashboard, you can preview the dashboard before actually creating and being charged.

> **Note**
>
> The dashboard created with Amazon CloudFormation in this solution displays metrics from the Region where the solution is deployed. Be sure to create the Amazon CloudFormation stack in the Region where your JVM metrics are published.
> If CloudWatch agent metrics are getting published to a different namespace than `CWAgent` (for example, if you've provided a customized namespace), you'll have to change the CloudFormation configuration to replace `CWAgent` with the customized namespace you are using.

To create the dashboard via CloudWatch Console

> **Note**
>
> Solution dashboards currently display garbage collection-related metrics only for the G1 Garbage Collector, which is the default collector for the latest Java versions. If you are using a different garbage collection algorithm, the widgets pertaining to garbage collection are empty. However, you can customize these widgets by changing the dashboard CloudFormation template and applying the appropriate garbage

collection type to the name dimension of the garbage collection-related metrics. For example, if you are using parallel garbage collection, change the **`name=\"G1 Young Generation\"`** to **`name=\"Parallel GC\"`** of the garbage collection count metric `jvm.gc.collections.count`.

1. Open the CloudWatch Console **Create Dashboard** using this link: https://console.aws.amazon.com/cloudwatch/home?#dashboards?dashboardTemplate=JvmOnEc2&referrer=os-catalog .
2. Verify that the selected Region on the console is the Region where the JVM workload is running.
3. Enter the name of the dashboard, then choose **Create Dashboard**.

   To easily differentiate this dashboard from similar dashboards in other Regions, we recommend including the Region name in the dashboard name, such as **`JVMDashboard-us-east-1`**.
4. Preview the dashboard and choose **Save** to create the dashboard.

To create the dashboard via Amazon CloudFormation

1. Open the Amazon CloudFormation **Quick create stack** wizard using this link: https://console.amazonaws.cn/cloudformation/home?#/stacks/quickcreate?templateURL=https://aws-observability-solutions-prod-us-east-1.s3.us-east-1.amazonaws.com/JVM_EC2/CloudWatch/CFN/v1.0.0/dashboard-template-1.0.0.json .
2. Verify that the selected Region on the console is the Region where the JVM workload is running.
3. For **Stack name**, enter a name to identity this stack, such as **`JVMDashboardStack`**.
4. In the **Parameters** section, specify the name of the dashboard under the **DashboardName** parameter.

   To easily differentiate this dashboard from similar dashboards in other Regions, we recommend including the Region name in the dashboard name, such as **`JVMDashboard-us-east-1`**.
5. Acknowledge access capabilities for transforms under **Capabilities and transforms**. Note that CloudFormation doesn't add any IAM resources.
6. Review the settings, then choose **Create stack**.

7. After the stack status is **CREATE_COMPLETE**, choose the **Resources** tab under the created stack and then choose the link under **Physical ID** to go to the dashboard. You can also access the dashboard in the CloudWatch console by choosing **Dashboards** in the left navigation pane of the console, and finding the dashboard name under **Custom Dashboards**.

If you want to edit the template file to customize it for any purpose, you can use **Upload a template file** option under **Create Stack Wizard** to upload the edited template. For more information, see Creating a stack on Amazon CloudFormation console. You can use this link to download the template: https://aws-observability-solutions-prod-us-east-1.s3.us-east-1.amazonaws.com/JVM_EC2/CloudWatch/CFN/v1.0.0/dashboard-template-1.0.0.json.

> **Note**
>
> Solution dashboards currently display garbage collection-related metrics only for the G1 Garbage Collector, which is the default collector for the latest Java versions. If you are using a different garbage collection algorithm, the widgets pertaining to garbage collection are empty. However, you can customize these widgets by changing the dashboard CloudFormation template and applying the appropriate garbage collection type to the name dimension of the garbage collection-related metrics. For example, if you are using parallel garbage collection, change the **`name=\"G1 Young Generation\"`** to **`name=\"Parallel GC\"`** of the garbage collection count metric `jvm.gc.collections.count`.

Get started with the JVM dashboard

Here are a few tasks that you can try out with the new JVM dashboard. These tasks allow you to validate that the dashboard is working correctly and provide you some hands-on experience using it to monitor a JVM process group. As you try these out, you'll get familiar with navigating the dashboard and interpreting the visualized metrics.

Select a process group

Use the **JVM Process Group Name** dropdown list to select the process group that you want to monitor. The dashboard automatically updates to display metrics for the selected process group. If you have multiple Java applications or environments, each might be represented as a separate process group. Selecting the appropriate process group ensures that you're viewing metrics specific to the application or environment that you want to analyze.

Review memory usage

From the dashboard overview section, find the **Heap Memory Usage Percentage** and **Non-Heap Memory Usage Percentage** widgets. These show the percentage of heap and non-heap memory being used across all JVMs in the selected process group. A high percentage indicates potential memory pressure that could lead to performance issues or `OutOfMemoryError` exceptions. You can also drill down to heap usage by host under **Memory usage by host** to check the hosts with high usage.

Analyze threads and classes loaded

In the **Threads and Classes Loaded by Host** section, find the **Top 10 Threads Count** and **Top 10 Classes Loaded** widgets. Look for any JVMs with an abnormally high number of threads or classes compared to others. Too many threads can indicate thread leaks or excessive concurrency, while a large number of loaded classes could point to potential class loader leaks or inefficient dynamic class generation.

Identify garbage collection issues

In the **Garbage Collection** section, find the **Top 10 Garbage Collections Invocations Per Minute** and **Top 10 Garbage Collection Duration** widgets for the different garbage collector types: **Young**, **Concurrent**, and **Mixed**. Look for any JVMs that have an unusually high number of collections or long collection durations compared to others. This could indicate configuration issues or memory leaks.

CloudWatch solution: NGINX workload on Amazon EC2

This solution helps you configure out-of-the-box metric collection using CloudWatch agents for NGINX application running on EC2 instances. For general information about all CloudWatch observability solutions, see CloudWatch observability solutions.

Topics

Requirements

This solution is relevant for the following conditions:

- Supported versions: NGINX version 1.24
- Compute: Amazon EC2
- Supports up to 500 EC2 instances across all NGINX workloads in a given Amazon Web Services Region
- Latest version of CloudWatch agent
- Prometheus Exporter: nginxinc/nginx-prometheus-exporter (Apache 2.0 license)
- SSM agent installed on EC2 instance

> **Note**
>
> Amazon Systems Manager (SSM agent) is pre-installed on some Amazon Machine Images (AMIs) provided by Amazon and trusted third-parties. If the agent isn't installed, you can install it manually using the procedure for your operating system type.
>
> - Manually installing and uninstalling SSM Agent on EC2 instances for Linux
> - Manually installing and uninstalling SSM Agent on EC2 instances for macOS
> - Manually installing and uninstalling SSM Agent on EC2 instances for Windows Server

Benefits

The solution delivers NGINX monitoring, providing valuable insights for the following use cases:

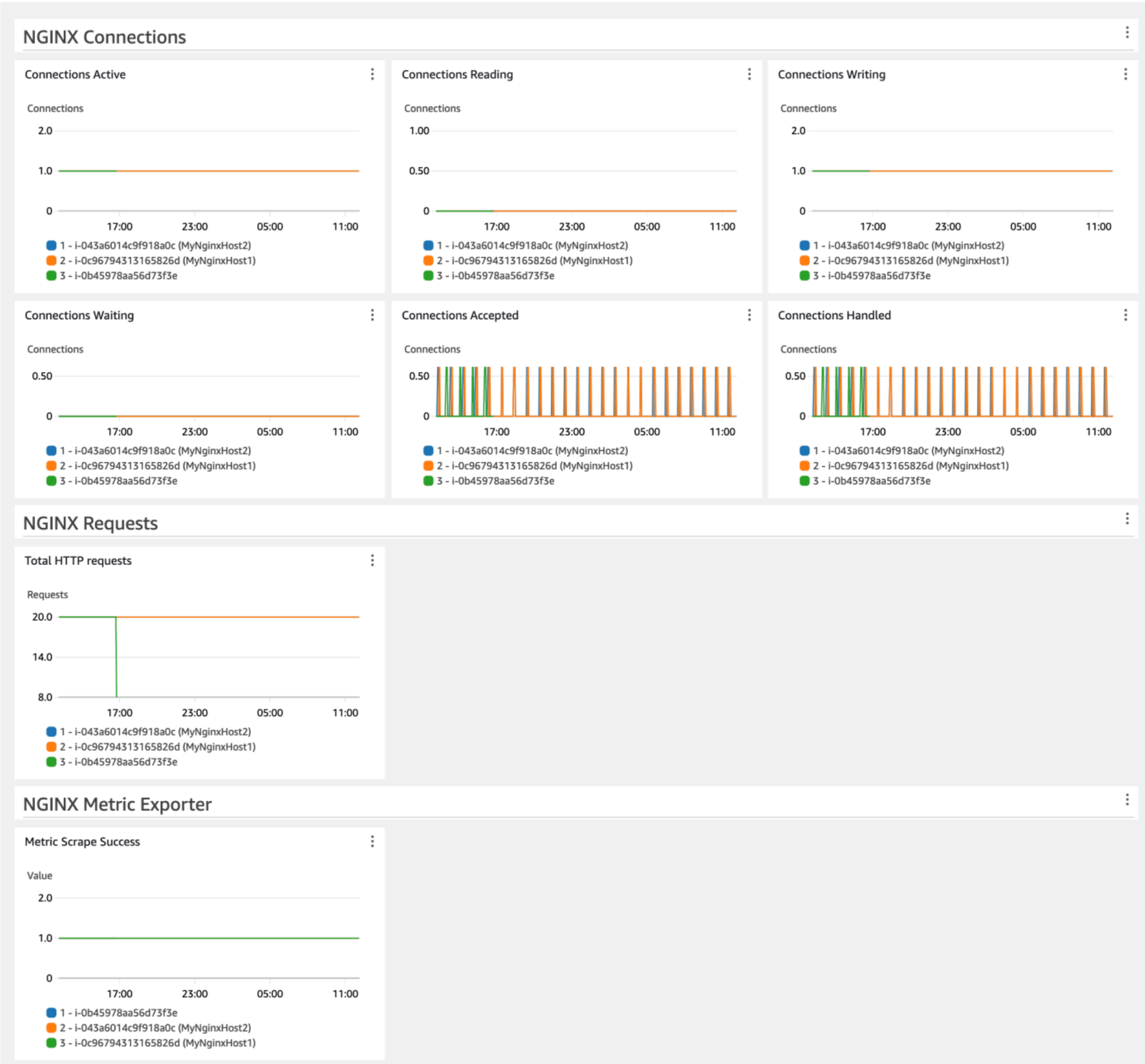
- Review connection metrics to identify potential bottlenecks, connection issues, or unexpected usage.
- Analyze HTTP request volume to understand overall traffic load on the NGINX.

Below are the key advantages of the solution:

- Automates metric collection for NGINX using CloudWatch agent configuration, eliminating manual instrumentation.

- Provides a pre-configured, consolidated CloudWatch dashboard for NGINX metrics. The dashboard will automatically handle metrics from new NGINX EC2 instances configured using the solution, even if those metrics don't exist when you first create the dashboard.

The following image is an example of the dashboard for this solution.



Costs

This solution creates and uses resources in your account. You are charged for standard usage, including the following:

- All metrics collected by the CloudWatch agent for this solution are published to CloudWatch Logs using the Embedded Metric Format (EMF). These CloudWatch logs are charged based on their volume and retention period. Therefore, you will not be billed for any **PutMetricData** API calls for this solution. The metrics extracted and ingested from your logs are charged as custom metrics. The number of metrics used by this solution depends on the number of EC2 hosts.
  - Each NGINX EC2 host configured for the solution publishes a total of eight metrics.
- One custom dashboard.

For more information about CloudWatch pricing, see [Amazon CloudWatch Pricing](#).

The pricing calculator can help you estimate approximate monthly costs for using this solution.

To use the pricing calculator to estimate your monthly solution costs

1. Open the [Amazon CloudWatch pricing calculator](#).
2. For **Choose a Region**, select the Amazon Web Services Region where you would like to deploy the solution.
3. In the **Metrics** section, for **Number of metrics**, enter **`8 * number of EC2 instances configured for this solution`**.
4. In the **Logs** section, for **Standard Logs: Data Ingested**, enter the estimated daily log volume generated by the CloudWatch Agent across all EC2 hosts. For example, five EC2 instances produce less than 1000 bytes per day. Once set up, you can check your byte usage using the `IncomingBytes` metric, vended by CloudWatch Logs. Be sure to select the appropriate log group.
5. In the **Logs** section, for **Log Storage/Archival (Standard and Vended Logs)**, select **`Yes to Store Logs: Assuming 1 month retention`**. Modify this value if you decide to make custom changes to the retention period.
6. In the **Dashboards and Alarms** section, for **Number of Dashboards**, enter **1**.
7. You can see your monthly estimated costs at the bottom of the pricing calculator.

CloudWatch agent configuration for this solution

The CloudWatch agent is software that runs continuously and autonomously on your servers and in containerized environments. It collects metrics, logs, and traces from your infrastructure and applications and sends them to CloudWatch and X-Ray.

For more information about the CloudWatch agent, see Collect metrics, logs, and traces with the CloudWatch agent.

The agent configuration in this solution collects a set of metrics to help you get started monitoring and observing your NGINX workload. The CloudWatch agent can be configured to collect more NGINX metrics than the dashboard displays by default. For a list of all NGINX metrics that you can collect, see Metrics for NGINX OSS.

Before configuring the CloudWatch agent, you must first configure NGINX to expose its metrics. Secondly, you must install and configure the third party Prometheus metric exporter.

Expose NGINX metrics

Note

The following commands are for Linux. Check NGINX for Windows page for equivalent commands in Windows Server.

You must first enable the `stub_status` module. Add a new location block in your NGINX configuration file. Add the following lines in the `server` block of your `nginx.conf` to enable NGINX's `stub_status` module:

```
location /nginx_status {
          stub_status on;
          allow 127.0.0.1; # Allow only localhost to access
          deny all; # Deny all other IPs
        }
```

Before reloading NGINX, validate your NGINX configuration:

```
sudo nginx -t
```

This validation command helps to prevent any unforeseen errors, which can cause your website to gown down. The following example demonstrates a successful response:

```
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful
```

Once you've successfully validated the updated configuration, reload NGINX (no output is expected):

```
sudo systemctl reload nginx
```

This command instructs the NGINX process to reload the configuration. Reloads are more graceful compared to a full restart. A reload starts the new worker process with a new configuration, gracefully shutting down old worker processes.

Test the NGINX status endpoint:

```
curl http://127.0.0.1/nginx_status
```

The following example demonstrates a successful response:

```
Active connections: 1
server accepts handled requests
6 6 6
Reading: 0 Writing: 1 Waiting: 0
```

The following example demonstrates a failure response (review the previous steps before proceeding):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/
xhtml11.dtd">
  <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
      <head>
          <title>The page is not found</title>
          ...
```

Configure Prometheus metric exporter

Download the latest NGINX Prometheus exporter release from the official GitHub repository. You must download the relevant binary for your platform.

The following example demonstrates commands for AMD64:

```
cd /tmp
wget https://github.com/nginxinc/nginx-prometheus-exporter/releases/download/v1.3.0/
nginx-prometheus-exporter_1.3.0_linux_amd64.tar.gz
tar -xzvf nginx-prometheus-exporter_1.3.0_linux_amd64.tar.gz
sudo cp nginx-prometheus-exporter /usr/local/bin/
rm /tmp/nginx-prometheus-exporter*
```

Run the Prometheus exporter and point it to the NGINX stub status page:

```
nohup /usr/local/bin/nginx-prometheus-exporter -nginx.scrape-uri http://127.0.0.1/
nginx_status &>/dev/null &
```

The following example demonstrates a response (background job ID and PID):

```
[1] 74699
```

Test the NGINX Prometheus endpoint

Validate that the NGINX Prometheus exporter has started to expose the relevant metrics:

```
curl http://localhost:port-number/metrics
```

The following example demonstrates a successful response:

```
# HELP go_gc_duration_seconds A summary of the pause duration of garbage collection
 cycles.
# TYPE go_gc_duration_seconds summary
go_gc_duration_seconds{quantile="0"} 0
go_gc_duration_seconds{quantile="0.25"} 0
...
# HELP nginx_connections_accepted Accepted client connections
# TYPE nginx_connections_accepted counter
nginx_connections_accepted 14
# HELP nginx_connections_active Active client connections
# TYPE nginx_connections_active gauge
nginx_connections_active 1
...
# TYPE promhttp_metric_handler_requests_total counter
promhttp_metric_handler_requests_total{code="200"} 1
```

```
promhttp_metric_handler_requests_total{code="500"} 0
promhttp_metric_handler_requests_total{code="503"} 0
```

Agent configuration for this solution

The metrics collected by the agent are defined in the agent configuration. The solution provides agent configurations to collect the recommended metrics with suitable dimensions for the solution's dashboard.

The steps for deploying the solution are described later in Deploy the agent for your solution. The following information is intended to help you understand how to customize the agent configuration for your environment.

You must customize some parts of the agent and Prometheus configurations for your environment such as the port number used by the Prometheus exporter.

The port used by the Prometheus exporter can be verified using the following command:

```
sudo netstat -antp | grep nginx-prom
```

The following example demonstrates a response (see port value 9113):

```
tcp6 0 0 :::9113 :::* LISTEN 76398/nginx-prometh
```

Agent configuration for NGINX hosts

The CloudWatch agent with Prometheus monitoring needs two configurations to scrape the Prometheus metrics. Each configuration will be stored as a separate parameter in SSM's Parameter Store, as detailed later in Step 2: Store the recommended CloudWatch agent configuration file in Systems Manager Parameter Store.

The first configuration is for the Prometheus exporter, as documented in Prometheus' scrape_config documentation. The second configuration is for the CloudWatch agent.

Prometheus configuration

Replace *port-number* with the port of your server.

```
global:
  scrape_interval: 30s
```

```
  scrape_timeout: 10s

scrape_configs:
- job_name: 'nginx'
  metrics_path: /metrics
  static_configs:
    - targets: ['localhost:port-number']
  ec2_sd_configs:
    - port: port-number
  relabel_configs:
    - source_labels: ['__meta_ec2_instance_id']
      target_label: InstanceId
  metric_relabel_configs:
    - source_labels: ['__name__']
      regex: 'nginx_up|nginx_http_requests_total|nginx_connections_.*'
      action: keep
```

CloudWatch agent configuration

As per the previous CloudWatch agent configuration, these metrics are published via CloudWatch Logs using the embedded metric format (EMF). These logs are configured to use the log group `nginx`. You can customize the *log_group_name* with a different name that represents the CloudWatch logs.

If you are using Windows Server, set *prometheus_config_path* in the following configuration to `C:\\ProgramData\\Amazon\\AmazonCloudWatchAgent\\prometheus.yaml`.

```
{
"agent": {
  "metrics_collection_interval": 60
},
"logs": {
  "metrics_collected": {
      "prometheus": {
          "log_group_name": "nginx",
          "prometheus_config_path": "/opt/aws/amazon-cloudwatch-agent/etc/
prometheus.yaml",
          "emf_processor": {
              "metric_declaration_dedup": true,
              "metric_namespace": "CWAgent",
              "metric_declaration":[
                  {
                      "source_labels":["InstanceId"],
```

```
                        "metric_selectors":["nginx_up", "nginx_http_requests_total",
 "nginx_connections*"],
                        "dimensions": [["InstanceId"]]
                    }
                ]
            }
        }
    }
}
}
```

Deploy the agent for your solution

There are several approaches for installing the CloudWatch agent, depending on the use case. We recommend using Systems Manager for this solution. It provides a console experience and makes it simpler to manage a fleet of managed servers within a single Amazon account. The instructions in this section use Systems Manager and are intended for when you don't have the CloudWatch agent running with existing configurations. You can check whether the CloudWatch agent is running by following the steps in Verify that the CloudWatch agent is running.

If you are already running the CloudWatch agent on the EC2 hosts where the workload is deployed and managing agent configurations, you can skip the instructions in this section and follow your existing deployment mechanism to update the configuration. Be sure to merge new CloudWatch agent and Prometheus configurations with your existing configurations, and then deploy the merged configurations. If you are using Systems Manager to store and manage the configuration for the CloudWatch agent, you can merge the configuration to the existing parameter value. For more information, see Managing CloudWatch agent configuration files.

> **Note**
>
> Using Systems Manager to deploy the following CloudWatch agent configurations will replace or overwrite any existing CloudWatch agent configuration on your EC2 instances. You can modify this configuration to suit your unique environment or use case. The metrics defined in configuration are the minimum required for the dashboard provided the solution.

The deployment process includes the following steps:

- Step 1: Ensure that the target EC2 instances have the required IAM permissions.

- Step 2: Store the recommended agent configuration file in the Systems Manager Parameter Store.
- Step 3: Install the CloudWatch agent on one or more EC2 instances using an Amazon CloudFormation stack.
- Step 4: Verify the agent setup is configured properly.

Step 1: Ensure the target EC2 instances have the required IAM permissions

You must grant permission for Systems Manager to install and configure the CloudWatch agent. You must grant permission for the CloudWatch agent to publish telemetry from your EC2 instance to CloudWatch. You must also grant the CloudWatch agent EC2 read access. EC2 read access is required for the EC2 InstanceId to be added as a metric dimension. This additional requirement is driven by `prometheus.yaml` as detailed above because it uses `__meta_ec2_instance_id` via EC2 Service Discovery.

Make sure that the IAM role attached to the instance has the **CloudWatchAgentServerPolicy**, **AmazonSSMManagedInstanceCore**, and **AmazonEC2ReadOnlyAccess** IAM policies attached.

- To create a role, see [Create IAM roles to use with the CloudWatch agent on Amazon EC2 instances](#).
- After the role is created, attach the role to your EC2 instances. To attach a role to an EC2 instance, follow the steps in [Attach an IAM role to an instance](#).

Step 2: Store the recommended CloudWatch agent configuration file in Systems Manager Parameter Store

Parameter Store simplifies the installation of the CloudWatch agent on an EC2 instance by securely storing and managing configuration parameters, eliminating the need for hard-coded values. This ensures a more secure and flexible deployment process, enabling centralized management and easier updates to configurations across multiple instances.

Use the following steps to store the recommended CloudWatch agent configuration file as a parameter in Parameter Store.

To create the CloudWatch agent configuration file as a parameter

1. Open the Amazon Systems Manager console at https://console.amazonaws.cn/systems-manager/.

2. Verify that the selected Region on the console is the Region where NGINX is running.
3. From the navigation pane, choose **Application Management**, **Parameter Store**
4. Follow these steps to create a new parameter for the configuration.

   a. Choose **Create parameter**.
   b. In the **Name** box, enter a name that you'll use to reference the CloudWatch agent configuration file in later steps. For example, **`AmazonCloudWatch-NGINX-CloudWatchAgent-Configuration`**.
   c. (Optional) In the **Description** box, type a description for the parameter.
   d. For **Parameter tier**, choose **Standard**.
   e. For **Type**, choose **String**.
   f. For **Data type**, choose **text**.
   g. In the **Value** box, paste the corresponding JSON block that was listed in [Agent configuration for NGINX hosts](). Be sure to customize as required. For example, the relevant `log_group_name`.
   h. Choose **Create parameter**.

To create the Prometheus configuration file as a parameter

1. Open the Amazon Systems Manager console at https://console.amazonaws.cn/systems-manager/.
2. From the navigation pane, choose **Application Management**, **Parameter Store**
3. Follow these steps to create a new parameter for the configuration.

   a. Choose **Create parameter**.
   b. In the **Name** box, enter a name that you'll use to reference the configuration file in later steps. For example, **`AmazonCloudWatch-NGINX-Prometheus-Configuration`**.
   c. (Optional) In the **Description** box, type a description for the parameter.
   d. For **Parameter tier**, choose **Standard**.
   e. For **Type**, choose **String**.
   f. For **Data type**, choose **text**.
   g. In the **Value** box, paste the corresponding YAML block that was listed in the [Agent configuration for NGINX hosts](). Be sure to customize as required. For example, the relevant port number as per `targets`.

h. Choose **Create parameter**.

Step 3: Install the CloudWatch agent and apply the configuration using an Amazon CloudFormation template

You can use Amazon CloudFormation to install the agent and configure it to use the CloudWatch agent configuration that you created in the previous steps.

To install and configure the CloudWatch agent for this solution

1. Open the Amazon CloudFormation **Quick create stack** wizard using this link: https://console.aws.amazon.com/cloudformation/home?#/stacks/quickcreate?templateURL=https://aws-observability-solutions-prod-us-east-1.s3.us-east-1.amazonaws.com/CloudWatchAgent/CFN/v1.0.0/cw-agent-installation-template-with-prometheus-config-1.0.0.json .
2. Verify that the selected Region on the console is the Region where the NGINX workload is running.
3. For **Stack name**, enter a name to identity this stack, such as **`CWAgentInstallationStack`**.
4. In the **Parameters** section, specify the following:
   a. For **CloudWatchAgentConfigSSM**, enter the name of the Amazon Systems Manager parameter for the agent configuration that you created earlier, such as **`AmazonCloudWatch-NGINX-CloudWatchAgent-Configuration`**.
   b. For **PrometheusConfigSSM**, enter the name of the Amazon Systems Manager parameter for the agent configuration that you created earlier, such as **`AmazonCloudWatch-NGINX-Prometheus-Configuration`**.
   c. To select the target instances, you have two options.
      i. For **InstanceIds**, specify a comma-delimited list of instance IDs list of instance IDs where you want to install the CloudWatch agent with this configuration. You can list a single instance or several instances.
      ii. If you are deploying at scale, you can specify the **TagKey** and the corresponding **TagValue** to target all EC2 instances with this tag and value. If you specify a **TagKey**, you must specify a corresponding **TagValue**. (For an Auto Scaling group, specify **`aws:autoscaling:groupName`** for the **TagKey** and specify the Auto Scaling group name for the **TagValue** to deploy to all instances within the Auto Scaling group.)
5. Review the settings, then choose **Create stack**.

If you want to edit the template file first to customize it, choose the **Upload a template file** option under **Create Stack Wizard** to upload the edited template. For more information, see Creating a stack on Amazon CloudFormation console. You can use the following link to download the template: https://aws-observability-solutions-prod-us-east-1.s3.us-east-1.amazonaws.com/CloudWatchAgent/CFN/v1.0.0/cw-agent-installation-template-with-prometheus-config-1.0.0.json.

> **Note**
>
> After this step is completed, this Systems Manager parameter will be associated with the CloudWatch agents running in the targeted instances. This means that:
>
> 1. If the Systems Manager parameter is deleted, the agent will stop.
> 2. If the Systems Manager parameter is edited, the configuration changes will automatically apply to the agent at the scheduled frequency which is 30 days by default.
> 3. If you want to immediately apply changes to this Systems Manager parameter, you must run this step again. For more information about associations, see Working with associations in Systems Manager.

Step 4: Verify the agent setup is configured properly

You can verify whether the CloudWatch agent is installed by following the steps in Verify that the CloudWatch agent is running. If the CloudWatch agent is not installed and running, make sure you have set up everything correctly.

- Be sure you have attached a role with correct permissions for the EC2 instance as described in Step 1: Ensure the target EC2 instances have the required IAM permissions.
- Be sure you have correctly configured the JSON for the Systems Manager parameter. Follow the steps in ???.

If everything is set up correctly, then you should see the NGINX metrics being published to CloudWatch. You can check the CloudWatch console to verify they are being published.

To verify that NGINX metrics are being published to CloudWatch

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. Choose **Metrics**, **All metrics**.

3. Make sure you've selected the Region where you deployed the solution, and choose **Custom namespaces**, **CWAgent**.
4. Search for metrics such as `nginx_http_requests_total`. If you see results for these metrics, then the metrics are being published to CloudWatch.

Create the NGINX solution dashboard

The dashboard provided by this solution presents NGINX workload metrics by aggregating and presenting metrics across all instances. The dashboard shows a breakdown of the top contributors (top 10 per metric widget) for each metric. This helps you to quickly identify outliers or instances that significantly contribute to the observed metrics.

To create the dashboard, you can use the following options:

- Use CloudWatch console to create the dashboard.
- Use Amazon CloudFormation console to deploy the dashboard.
- Download the Amazon CloudFormation infrastructure as code and integrate it as part of your continuous integration (CI) automation.

By using the CloudWatch console to create a dashboard, you can preview the dashboard before actually creating and being charged.

> **Note**
>
> The dashboard created with Amazon CloudFormation in this solution displays metrics from the Region where the solution is deployed. Be sure to create the Amazon CloudFormation stack in the Region where your NGINX metrics are published.
> If you've specified a custom namespace other than `CWAgent` in the CloudWatch agent configuration, you'll have to change the CloudFormation template for the dashboard to replace `CWAgent` with the customized namespace you are using.

To create the dashboard via CloudWatch Console

1. Open the CloudWatch Console **Create Dashboard** using this link: https://console.aws.amazon.com/cloudwatch/home?#dashboards?dashboardTemplate=NginxOnEc2&referrer=os-catalog .

2. Verify that the selected Region on the console is the Region where the NGINX workload is running.
3. Enter the name of the dashboard, then choose **Create Dashboard**.

   To easily differentiate this dashboard from similar dashboards in other Regions, we recommend including the Region name in the dashboard name, such as **`NGINXDashboard-us-east-1`**.
4. Preview the dashboard and choose **Save** to create the dashboard.

To create the dashboard via Amazon CloudFormation

1. Open the Amazon CloudFormation **Quick create stack** wizard using this link: https://console.aws.amazon.com/cloudformation/home?#/stacks/quickcreate?templateURL=https://aws-observability-solutions-prod-us-east-1.s3.us-east-1.amazonaws.com/NGINX_EC2/CloudWatch/CFN/v1.0.0/dashboard-template-1.0.0.json .
2. Verify that the selected Region on the console is the Region where the NGINX workload is running.
3. For **Stack name**, enter a name to identity this stack, such as **`NGINXDashboardStack`**.
4. In the **Parameters** section, specify the name of the dashboard under the **DashboardName** parameter.
5. To easily differentiate this dashboard from similar dashboards in other Regions, we recommend including the Region name in the dashboard name, such as **`NGINXDashboard-us-east-1`**.
6. Acknowledge access capabilities for transforms under **Capabilities and transforms**. Note that CloudFormation doesn't add any IAM resources.
7. Review the settings, then choose **Create stack**.
8. After the stack status is **CREATE_COMPLETE**, choose the **Resources** tab under the created stack and then choose the link under **Physical ID** to go to the dashboard. You can also access the dashboard in the CloudWatch console by choosing **Dashboards** in the left navigation pane of the console, and finding the dashboard name under **Custom Dashboards**.

If you want to edit the template file first to customize it, choose the **Upload a template file** option under **Create Stack Wizard** to upload the edited template. For more information, see Creating a stack on Amazon CloudFormation console. You can use the following link to download the template: https://aws-observability-solutions-prod-us-east-1.s3.us-east-1.amazonaws.com/NGINX_EC2/CloudWatch/CFN/v1.0.0/dashboard-template-1.0.0.json.

Get started with the NGINX dashboard

Here are a few tasks that you can try out with the new NGINX dashboard. These tasks allow you to validate that the dashboard is working correctly and provide you some hands-on experience using it to monitor a NGINX workload. As you try these out, you'll get familiar with navigating the dashboard and interpreting the visualized metrics.

Review connection metrics

In the **Connections** section, you can find several key metrics that provide insights into the client connection handling of your NGINX server. Monitoring these connection metrics can help you identify potential bottlenecks, connection issues, or unexpected connection patterns.

- Accepted client connections
- Active client connections
- Handled client connections
- Connections reading requests
- Idle client connections
- Connections writing responses

Analyze HTTP request volume

The `request` metric in the **HTTP Requests** section shows the total number of HTTP requests handled by the NGINX server. Tracking this metric over time can help you understand the overall traffic load on your NGINX infrastructure and plan for resource allocation and scaling accordingly.

CloudWatch solution: NVIDIA GPU workload on Amazon EC2

This solution helps you configure out-of-the-box metric collection using CloudWatch agents for NVIDIA GPU workloads running on EC2 instances. Additionally, it helps you set up a pre-configured CloudWatch dashboard. For general information about all CloudWatch observability solutions, see CloudWatch observability solutions.

Topics

- Requirements

- Benefits
- CloudWatch agent configuration for this solution
- Deploy the agent for your solution
- Create the NVIDIA GPU solution dashboard

Requirements

This solution is relevant for the following conditions:

- Compute: Amazon EC2
- Supports up to 500 GPUs across all EC2 instances in a given Amazon Web Services Region
- Latest version of CloudWatch agent
- SSM agent installed on EC2 instance
- The EC2 instance must have an NVIDIA driver installed. NVIDIA drivers are pre-installed on some Amazon Machine Images (AMIs). Otherwise, you can manually install the driver. For more information, see Install NVIDIA drivers on Linux instances.

> **Note**
>
> Amazon Systems Manager (SSM agent) is pre-installed on some Amazon Machine Images (AMIs) provided by Amazon and trusted third-parties. If the agent isn't installed, you can install it manually using the procedure for your operating system type.
>
> - Manually installing and uninstalling SSM Agent on EC2 instances for Linux
> - Manually installing and uninstalling SSM Agent on EC2 instances for macOS
> - Manually installing and uninstalling SSM Agent on EC2 instances for Windows Server

Benefits

The solution delivers NVIDIA monitoring, providing valuable insights for the following use cases:

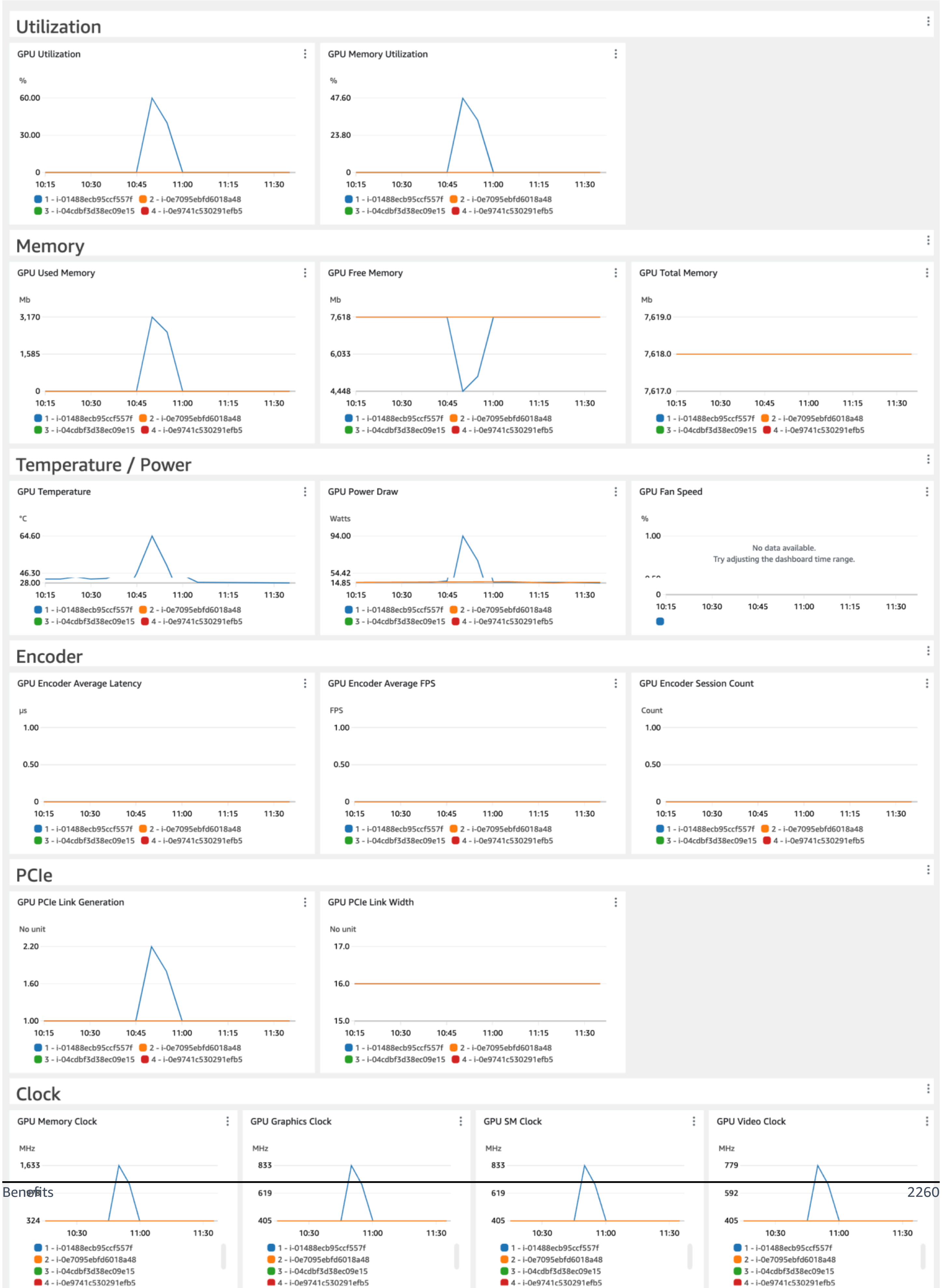
- Analyze GPU and memory usage for performance bottlenecks or the need for additional resources.
- Monitor temperature and power draw to ensure GPUs operate within safe limits.

- Evaluate encoder performance for GPU video workloads.
- Verify PCIe connectivity for expected generation and width.
- Monitor GPU clock speeds to detect scaling and throttling issues.

Below are the key advantages of the solution:

- Automates metric collection for NVIDIA using CloudWatch agent configuration, eliminating manual instrumentation.
- Provides a pre-configured, consolidated CloudWatch dashboard for NVIDIA metrics. The dashboard will automatically handle metrics from new NVIDIA EC2 instances configured using the solution, even if those metrics don't exist when you first create the dashboard.

The following image is an example of the dashboard for this solution.



Utilization
GPU Utilization
%
60.00
30.00
0
10:15 10:30 10:45 11:00 11:15 11:30
1 - i-01488ecb95ccf557f
2 - i-0e7095ebfd6018a48
3 - i-04cdbf3d38ec09e15
4 - i-0e9741c530291efb5
GPU Memory Utilization
%
47.60
23.80
0
Memory
GPU Used Memory
Mb
3,170
1,585
0
GPU Free Memory
Mb
7,618
6,033
4,448
GPU Total Memory
Mb
7,619.0
7,618.0
7,617.0
Temperature / Power
GPU Temperature
°C
64.60
46.30
28.00
GPU Power Draw
Watts
94.00
54.42
14.85
GPU Fan Speed
%
1.00
No data available.
Try adjusting the dashboard time range.
0
Encoder
GPU Encoder Average Latency
μs
1.00
0.50
0
GPU Encoder Average FPS
FPS
GPU Encoder Session Count
Count
PCIe
GPU PCIe Link Generation
No unit
2.20
1.60
1.00
GPU PCIe Link Width
No unit
17.0
16.0
15.0
Clock
GPU Memory Clock
MHz
1,633
324
GPU Graphics Clock
MHz
833
619
405
GPU SM Clock
MHz
833
619
405
GPU Video Clock
MHz
779
592
405
10:30 11:00 11:30

Costs

This solution creates and uses resources in your account. You are charged for standard usage, including the following:

- All metrics collected by the CloudWatch agent are charged as custom metrics. The number of metrics used by this solution depends on the number of EC2 hosts.
  - Each EC2 host configured for the solution publishes a total of 17 metrics per GPU.
- One custom dashboard.
- API operations requested by the CloudWatch agent to publish the metrics. With the default configuration for this solution, the CloudWatch agent calls the **PutMetricData** once every minute for each EC2 host. This means the **PutMetricData** API will be called `30*24*60=43,200` in a 30-day month for each EC2 host.

For more information about CloudWatch pricing, see [Amazon CloudWatch Pricing](https://aws.amazon.com/cloudwatch/pricing/).

The pricing calculator can help you estimate approximate monthly costs for using this solution.

To use the pricing calculator to estimate your monthly solution costs

1. Open the [Amazon CloudWatch pricing calculator](https://calculator.aws/#/createCalculator/CloudWatch).
2. For **Choose a Region**, select the Region where you would like to deploy the solution.
3. In the **Metrics** section, for **Number of metrics**, enter **`17 * average number of GPUs per EC2 host * number of EC2 instances configured for this solution`**.
4. In the **APIs** section, for **Number of API requests**, enter **`43200 * number of EC2 instances configured for this solution`**.
5. By default, the CloudWatch agent performs one **PutMetricData** operation each minute for each EC2 host.
6. In the **Dashboards and Alarms** section, for **Number of Dashboards**, enter **1**.
7. You can see your monthly estimated costs at the bottom of the pricing calculator.

CloudWatch agent configuration for this solution

The CloudWatch agent is software that runs continuously and autonomously on your servers and in containerized environments. It collects metrics, logs, and traces from your infrastructure and applications and sends them to CloudWatch and X-Ray.

For more information about the CloudWatch agent, see Collect metrics, logs, and traces with the CloudWatch agent.

The agent configuration in this solution collects a set of metrics to help you get started monitoring and observing your NVIDIA GPU. The CloudWatch agent can be configured to collect more NVIDIA GPU metrics than the dashboard displays by default. For a list of all NVIDIA GPU metrics that you can collect, see Collect NVIDIA GPU metrics.

Agent configuration for this solution

The metrics collected by the agent are defined in the agent configuration. The solution provides agent configurations to collect the recommended metrics with suitable dimensions for the solution's dashboard.

Use the following CloudWatch agent configuration on EC2 instances with NVIDIA GPUs. Configuration will be stored as a parameter in SSM's Parameter Store, as detailed later in Step 2: Store the recommended CloudWatch agent configuration file in Systems Manager Parameter Store.

```
{
    "metrics": {
        "namespace": "CWAgent",
        "append_dimensions": {
            "InstanceId": "${aws:InstanceId}"
        },
        "metrics_collected": {
            "nvidia_gpu": {
                "measurement": [
                    "utilization_gpu",
                    "temperature_gpu",
                    "power_draw",
                    "utilization_memory",
                    "fan_speed",
                    "memory_total",
                    "memory_used",
                    "memory_free",
                    "pcie_link_gen_current",
                    "pcie_link_width_current",
                    "encoder_stats_session_count",
                    "encoder_stats_average_fps",
                    "encoder_stats_average_latency",
                    "clocks_current_graphics",
                    "clocks_current_sm",
```

```
                    "clocks_current_memory",
                    "clocks_current_video"
                ],
                "metrics_collection_interval": 60
            }
        }
    },
    "force_flush_interval": 60
}
```

Deploy the agent for your solution

There are several approaches for installing the CloudWatch agent, depending on the use case. We recommend using Systems Manager for this solution. It provides a console experience and makes it simpler to manage a fleet of managed servers within a single Amazon account. The instructions in this section use Systems Manager and are intended for when you don't have the CloudWatch agent running with existing configurations. You can check whether the CloudWatch agent is running by following the steps in Verify that the CloudWatch agent is running.

If you are already running the CloudWatch agent on the EC2 hosts where the workload is deployed and managing agent configurations, you can skip the instructions in this section and follow your existing deployment mechanism to update the configuration. Be sure to merge the agent configuration of NVIDIA GPU with your existing agent configuration, and then deploy the merged configuration. If you are using Systems Manager to store and manage the configuration for the CloudWatch agent, you can merge the configuration to the existing parameter value. For more information, see Managing CloudWatch agent configuration files.

> **Note**
>
> Using Systems Manager to deploy the following CloudWatch agent configurations will replace or overwrite any existing CloudWatch agent configuration on your EC2 instances. You can modify this configuration to suit your unique environment or use case. The metrics defined in configuration are the minimum required for the dashboard provided the solution.

The deployment process includes the following steps:

- Step 1: Ensure that the target EC2 instances have the required IAM permissions.

- Step 2: Store the recommended agent configuration file in the Systems Manager Parameter Store.
- Step 3: Install the CloudWatch agent on one or more EC2 instances using an Amazon CloudFormation stack.
- Step 4: Verify the agent setup is configured properly.

Step 1: Ensure the target EC2 instances have the required IAM permissions

You must grant permission for Systems Manager to install and configure the CloudWatch agent. You must also grant permission for the CloudWatch agent to publish telemetry from your EC2 instance to CloudWatch. Make sure that the IAM role attached to the instance has the **CloudWatchAgentServerPolicy** and **AmazonSSMManagedInstanceCore** IAM policies attached.

- To create a role, see [Create IAM roles to use with the CloudWatch agent on Amazon EC2 instances](#).
- After the role is created, attach the role to your EC2 instances. To attach a role to an EC2 instance, follow the steps in [Attach an IAM role to an instance](#).

Step 2: Store the recommended CloudWatch agent configuration file in Systems Manager Parameter Store

Parameter Store simplifies the installation of the CloudWatch agent on an EC2 instance by securely storing and managing configuration parameters, eliminating the need for hard-coded values. This ensures a more secure and flexible deployment process, enabling centralized management and easier updates to configurations across multiple instances.

Use the following steps to store the recommended CloudWatch agent configuration file as a parameter in Parameter Store.

To create the CloudWatch agent configuration file as a parameter

1. Open the Amazon Systems Manager console at https://console.amazonaws.cn/systems-manager/.
2. Verify that the selected Region on the console is the Region where the NVIDIA GPU workload is running.
3. From the navigation pane, choose **Application Management**, **Parameter Store**.

4. Follow these steps to create a new parameter for the configuration.

   a. Choose **Create parameter**.

   b. In the **Name** box, enter a name that you'll use to reference the CloudWatch agent configuration file in later steps. For example, **`AmazonCloudWatch-NVIDIA-GPU-Configuration`**.

   c. (Optional) In the **Description** box, type a description for the parameter.

   d. For **Parameter tier**, choose **Standard**.

   e. For **Type**, choose **String**.

   f. For **Data type**, choose **text**.

   g. In the **Value** box, paste the corresponding JSON block that was listed in Agent configuration for this solution.

   h. Choose **Create parameter**.

Step 3: Install the CloudWatch agent and apply the configuration using an Amazon CloudFormation template

You can use Amazon CloudFormation to install the agent and configure it to use the CloudWatch agent configuration that you created in the previous steps.

To install and configure the CloudWatch agent for this solution

1. Open the Amazon CloudFormation **Quick create stack** wizard using this link: https://console.aws.amazon.com/cloudformation/home?#/stacks/quickcreate?templateURL=https://aws-observability-solutions-prod-us-east-1.s3.us-east-1.amazonaws.com/CloudWatchAgent/CFN/v1.0.0/cw-agent-installation-template-1.0.0.json.

2. Verify that the selected Region on the console is the Region where the NVIDIA GPU workload is running.

3. For **Stack name**, enter a name to identity this stack, such as **`CWAgentInstallationStack`**.

4. In the **Parameters** section, specify the following:

   a. For **CloudWatchAgentConfigSSM**, enter the name of the Systems Manager parameter for the agent configuration that you created earlier, such as **`AmazonCloudWatch-NVIDIA-GPU-Configuration`**.

   b. To select the target instances, you have two options.

i. For **InstanceIds**, specify a comma-delimited list of instance IDs list of instance IDs where you want to install the CloudWatch agent with this configuration. You can list a single instance or several instances.

ii. If you are deploying at scale, you can specify the **TagKey** and the corresponding **TagValue** to target all EC2 instances with this tag and value. If you specify a **TagKey**, you must specify a corresponding **TagValue**. (For an Auto Scaling group, specify **`aws:autoscaling:groupName`** for the **TagKey** and specify the Auto Scaling group name for the **TagValue** to deploy to all instances within the Auto Scaling group.)

5. Review the settings, then choose **Create stack**.

If you want to edit the template file first to customize it, choose the **Upload a template file** option under **Create Stack Wizard** to upload the edited template. For more information, see [Creating a stack on Amazon CloudFormation console].

> **Note**
>
> After this step is completed, this Systems Manager parameter will be associated with the CloudWatch agents running in the targeted instances. This means that:
>
> 1. If the Systems Manager parameter is deleted, the agent will stop.
> 2. If the Systems Manager parameter is edited, the configuration changes will automatically apply to the agent at the scheduled frequency which is 30 days by default.
> 3. If you want to immediately apply changes to this Systems Manager parameter, you must run this step again. For more information about associations, see [Working with associations in Systems Manager].

Step 4: Verify the agent setup is configured properly

You can verify whether the CloudWatch agent is installed by following the steps in [Verify that the CloudWatch agent is running]. If the CloudWatch agent is not installed and running, make sure you have set up everything correctly.

- Be sure you have attached a role with correct permissions for the EC2 instance as described in [Step 1: Ensure the target EC2 instances have the required IAM permissions].

- Be sure you have correctly configured the JSON for the Systems Manager parameter. Follow the steps in ???.

If everything is set up correctly, then you should see the NVIDIA GPU metrics being published to CloudWatch. You can check the CloudWatch console to verify they are being published.

To verify that NVIDIA GPU metrics are being published to CloudWatch

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. Choose **Metrics**, **All metrics**.
3. Make sure you've selected the Region where you deployed the solution, and choose **Custom namespaces**, **CWAgent**.
4. Search for the metrics mentioned in Agent configuration for this solution, such as `nvidia_smi_utilization_gpu`. If you see results for these metrics, then the metrics are being published to CloudWatch.

Create the NVIDIA GPU solution dashboard

The dashboard provided by this solution presents NVIDIA GPUs metrics by aggregating and presenting metrics across all instances. The dashboard shows a breakdown of the top contributors (top 10 per metric widget) for each metric. This helps you to quickly identify outliers or instances that significantly contribute to the observed metrics.

To create the dashboard, you can use the following options:

- Use CloudWatch console to create the dashboard.
- Use Amazon CloudFormation console to deploy the dashboard.
- Download the Amazon CloudFormation infrastructure as code and integrate it as part of your continuous integration (CI) automation.

By using the CloudWatch console to create a dashboard, you can preview the dashboard before actually creating and being charged.

> **Note**
>
> The dashboard created with Amazon CloudFormation in this solution displays metrics from the Region where the solution is deployed. Be sure to create the Amazon CloudFormation stack in the Region where your NVIDIA GPU metrics are published.
> If you've specified a custom namespace other than CWAgent in the CloudWatch agent configuration, you'll have to change the Amazon CloudFormation template for the dashboard to replace CWAgent with the customized namespace you are using.

To create the dashboard via CloudWatch Console

1. Open the CloudWatch Console **Create Dashboard** using this link: https://console.aws.amazon.com/cloudwatch/home?#dashboards?dashboardTemplate=NvidiaGpuOnEc2&referrer=os-catalog .
2. Verify that the selected Region on the console is the Region where the NVIDIA GPU workload is running.
3. Enter the name of the dashboard, then choose **Create Dashboard**.

   To easily differentiate this dashboard from similar dashboards in other Regions, we recommend including the Region name in the dashboard name, such as **`NVIDIA-GPU-Dashboard-us-east-1`**.
4. Preview the dashboard and choose **Save** to create the dashboard.

To create the dashboard via Amazon CloudFormation

1. Open the Amazon CloudFormation **Quick create stack** wizard using this link: https://console.aws.amazon.com/cloudformation/home?#/stacks/quickcreate?templateURL=https://aws-observability-solutions-prod-us-east-1.s3.us-east-1.amazonaws.com/NVIDIA_GPU_EC2/CloudWatch/CFN/v1.0.0/dashboard-template-1.0.0.json.
2. Verify that the selected Region on the console is the Region where the NVIDIA GPU workload is running.
3. For **Stack name**, enter a name to identity this stack, such as **`NVIDIA-GPU-DashboardStack`**.
4. In the **Parameters** section, specify the name of the dashboard under the **DashboardName** parameter.

5. To easily differentiate this dashboard from similar dashboards in other Regions, we recommend including the Region name in the dashboard name, such as **`NVIDIA-GPU-Dashboard-us-east-1`**.
6. Acknowledge access capabilities for transforms under **Capabilities and transforms**. Note that Amazon CloudFormation doesn't add any IAM resources.
7. Review the settings, then choose **Create stack**.
8. After the stack status is **CREATE_COMPLETE**, choose the **Resources** tab under the created stack and then choose the link under **Physical ID** to go to the dashboard. You can also access the dashboard in the CloudWatch console by choosing **Dashboards** in the left navigation pane of the console, and finding the dashboard name under **Custom Dashboards**.

If you want to edit the template file to customize it for any purpose, you can use **Upload a template file** option under **Create Stack Wizard** to upload the edited template. For more information, see Creating a stack on Amazon CloudFormation console. You can use this link to download the template: https://aws-observability-solutions-prod-us-east-1.s3.us-east-1.amazonaws.com/NVIDIA_GPU_EC2/CloudWatch/CFN/v1.0.0/dashboard-template-1.0.0.json .

Get started with the NVIDIA GPU dashboard

Here are a few tasks that you can try out with the new NVIDIA GPU dashboard. These tasks allow you to validate that the dashboard is working correctly and provide you some hands-on experience using it to monitor your NVIDIA GPUs. As you try these out, you'll get familiar with navigating the dashboard and interpreting the visualized metrics.

Review GPU utilization

From the **Utilization** section, find the **GPU Utilization** and **Memory Utilization** widgets. These show the percentage of time the GPU is being actively used for computations and the percentage of global memory being read or written, respectively. High utilization could indicate potential performance bottlenecks or the need for additional GPU resources.

Analyze GPU memory usage

In the **Memory** section, find the **Total Memory**, **Used Memory**, and **Free Memory** widgets. These provide insights into the overall memory capacity of the GPUs and how much memory is currently being consumed or available. Memory pressure could lead to performance issues or out-of-memory errors, so it's important to monitor these metrics and ensure sufficient memory is available for your workloads.

Monitor temperature and power draw

In the **Temperature / Power** section, find the **GPU Temperature** and **Power Draw** widgets. These metrics are essential for ensuring that your GPUs are operating within safe thermal and power limits.

Identify encoder performance

In the **Encoder** section, find the **Encoder Session Count**, **Average FPS**, and **Average Latency** widgets. These metrics are relevant if you're running video encoding workloads on your GPUs. Monitor these metrics to ensure that your encoders are performing optimally and identify any potential bottlenecks or performance issues.

Check PCIe link status

In the **PCIe** section, find the **PCIe Link Generation** and **PCIe Link Width** widgets. These metrics provide information about the PCIe link connecting the GPU to the host system. Ensure that the link is operating at the expected generation and width to avoid potential performance limitations due to PCIe bottlenecks.

Review GPU clocks

In the **Clock** section, find the **Graphics Clock**, **SM Clock**, **Memory Clock**, and **Video Clock** widgets. These metrics show the current operating frequencies of various GPU components. Monitoring these clocks can help identify potential issues with GPU clock scaling or frequency throttling, which could impact performance.

CloudWatch solution: Kafka workload on Amazon EC2

This solution helps you configure out-of-the-box metric collection using CloudWatch agents for Kafka workloads (brokers, producers, and consumers) running on EC2 instances. Additionally, it helps you set up a pre-configured CloudWatch dashboard. For general information about all CloudWatch observability solutions, see CloudWatch observability solutions.

Topics

Requirements

This solution is relevant for the following conditions:

- Workload: Kafka v0.8.2.x and later
- Compute: Amazon EC2
- Supports up to 500 EC2 instances across all Kafka workloads in a given Amazon Web Services Region
- Latest version of CloudWatch agent
- SSM agent installed on EC2 instance

> **Note**
>
> Amazon Systems Manager (SSM agent) is pre-installed on some Amazon Machine Images (AMIs) provided by Amazon and trusted third-parties. If the agent isn't installed, you can install it manually using the procedure for your operating system type.
>
> - Manually installing and uninstalling SSM Agent on EC2 instances for Linux
> - Manually installing and uninstalling SSM Agent on EC2 instances for macOS
> - Manually installing and uninstalling SSM Agent on EC2 instances for Windows Server

Benefits

The solution delivers Kafka server monitoring, providing valuable insights for the following use cases:

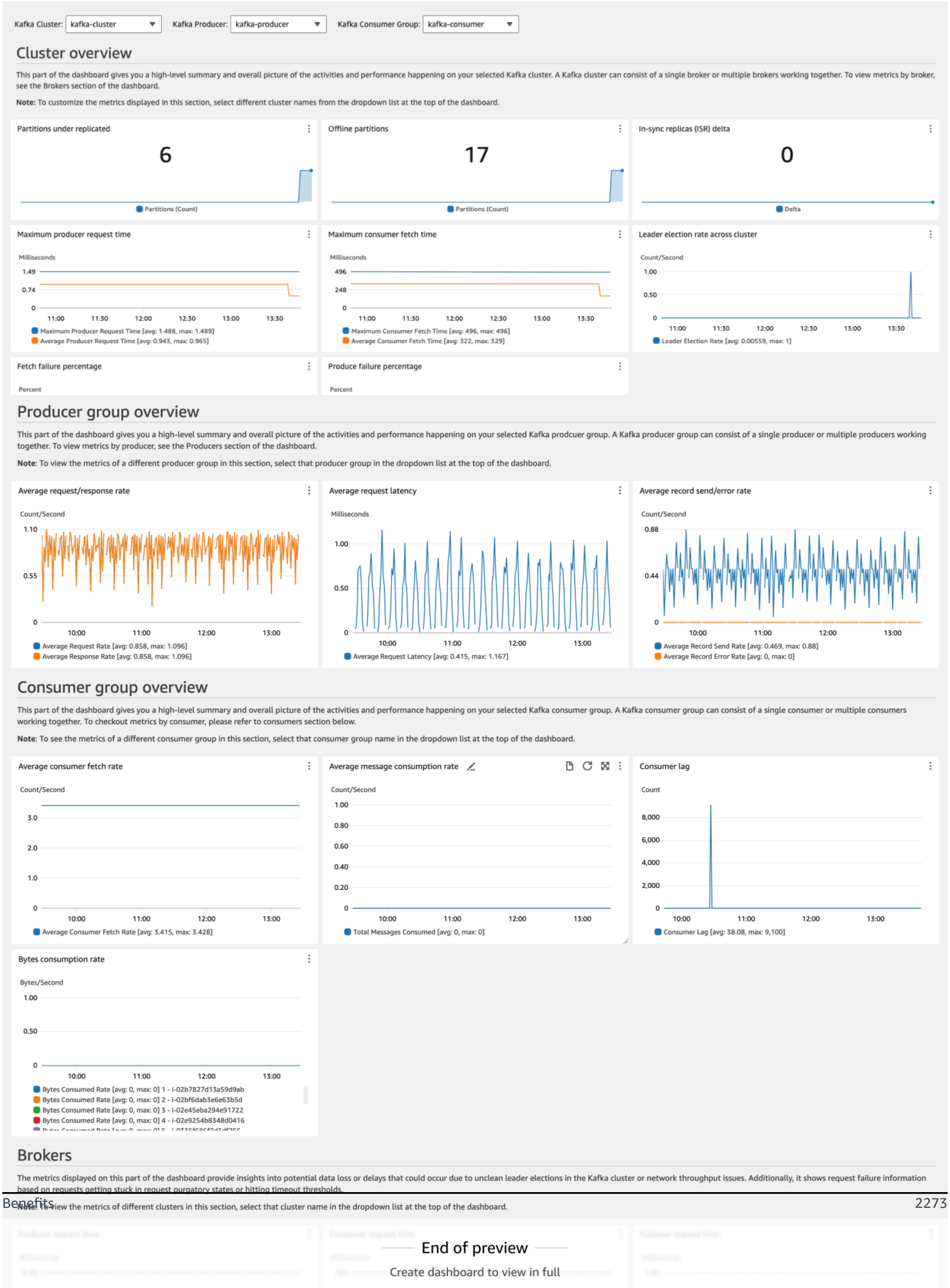
- Monitor Kafka cluster health via replication and sync metrics.
- Track broker performance through request failures and latencies along with network traffic.
- Monitor producer/consumer errors, latencies, and consumer lag.

- Analyze underlying JVM performance for Kafka clusters.
- Switch between multiple Kafka clusters, producers, and consumers configured via the solution under the same account.

Below are the key advantages of the solution:

- Automates metric collection for Kafka and the underlying JVM using CloudWatch agent configuration, eliminating manual instrumentation.
- Provides a pre-configured, consolidated CloudWatch dashboard for Kafka and JVM metrics. The dashboard will automatically handle metrics from new Kafka EC2 instances configured using the solution, even if those metrics don't exist when you first create the dashboard. It also allows you to group the metrics into logical applications for easier focus and management.

The following image is an example of the dashboard for this solution.


Kafka Cluster: kafka-cluster
Kafka Producer: kafka-producer
Kafka Consumer Group: kafka-consumer
Cluster overview
This part of the dashboard gives you a high-level summary and overall picture of the activities and performance happening on your selected Kafka cluster. A Kafka cluster can consist of a single broker or multiple brokers working together. To view metrics by broker, see the Brokers section of the dashboard.
Note: To customize the metrics displayed in this section, select different cluster names from the dropdown list at the top of the dashboard.
Partitions under replicated
6
Partitions (Count)
Offline partitions
17
Partitions (Count)
In-sync replicas (ISR) delta
0
Delta
Maximum producer request time
Milliseconds
Maximum Producer Request Time [avg: 1.488, max: 1.489]
Average Producer Request Time [avg: 0.943, max: 0.965]
Maximum consumer fetch time
Milliseconds
Maximum Consumer Fetch Time [avg: 496, max: 496]
Average Consumer Fetch Time [avg: 322, max: 329]
Leader election rate across cluster
Count/Second
Leader Election Rate [avg: 0.00559, max: 1]
Fetch failure percentage
Percent
Produce failure percentage
Percent
Producer group overview
This part of the dashboard gives you a high-level summary and overall picture of the activities and performance happening on your selected Kafka prodcuer group. A Kafka producer group can consist of a single producer or multiple producers working together. To view metrics by producer, see the Producers section of the dashboard.
Note: To view the metrics of a different producer group in this section, select that producer group in the dropdown list at the top of the dashboard.
Average request/response rate
Count/Second
Average Request Rate [avg: 0.858, max: 1.096]
Average Response Rate [avg: 0.858, max: 1.096]
Average request latency
Milliseconds
Average Request Latency [avg: 0.415, max: 1.167]
Average record send/error rate
Count/Second
Average Record Send Rate [avg: 0.469, max: 0.88]
Average Record Error Rate [avg: 0, max: 0]
Consumer group overview
This part of the dashboard gives you a high-level summary and overall picture of the activities and performance happening on your selected Kafka consumer group. A Kafka consumer group can consist of a single consumer or multiple consumers working together. To checkout metrics by consumer, please refer to consumers section below.
Note: To see the metrics of a different consumer group in this section, select that consumer group name in the dropdown list at the top of the dashboard.
Average consumer fetch rate
Count/Second
Average Consumer Fetch Rate [avg: 3.415, max: 3.428]
Average message consumption rate
Count/Second
Total Messages Consumed [avg: 0, max: 0]
Consumer lag
Count
Consumer Lag [avg: 38.08, max: 9,100]
Bytes consumption rate
Bytes/Second
Bytes Consumed Rate [avg: 0, max: 0] 1 - i-02b7827d13a59d9ab
Bytes Consumed Rate [avg: 0, max: 0] 2 - i-02bf6dab3e6e63b5d
Bytes Consumed Rate [avg: 0, max: 0] 3 - i-02e45eba294e91722
Bytes Consumed Rate [avg: 0, max: 0] 4 - i-02e9254b8348d0416
Brokers
The metrics displayed on this part of the dashboard provide insights into potential data loss or delays that could occur due to unclean leader elections in the Kafka cluster or network throughput issues. Additionally, it shows request failure information based on requests getting stuck in request purgatory states or hitting timeout thresholds.
Note: To view the metrics of different clusters in this section, select that cluster name in the dropdown list at the top of the dashboard.
End of preview
Create dashboard to view in full

Costs

This solution creates and uses resources in your account. You are charged for standard usage, including the following:

- All metrics collected by the CloudWatch agent are charged as custom metrics. The number of metrics used by this solution depends on the number of EC2 hosts.
  - Each broker host configured for the solution publishes 33 metrics plus one metric (`disk_used_percent`) for which the metric count for each EC2 host depends on the number of disk paths for that host.
  - Each producer host configured for the solution publishes three metrics with the `topic` dimension and three metrics without the `topic` dimension. For the metrics with the `topic` dimension, each topic counts as a separate metric.
  - Each consumer host configured for the solution publishes two metrics with `topic` dimensions and three metrics without `topic` dimensions. For the metrics with topic dimensions, each topic counts as a separate metric.
- One custom dashboard.
- API operations requested by the CloudWatch agent to publish the metrics. With the default configuration for this solution, the CloudWatch agent calls the **PutMetricData** once every minute for each EC2 host. This means the **PutMetricData** API will be called `30*24*60=43,200` in a 30-day month for each EC2 host.

For more information about CloudWatch pricing, see [Amazon CloudWatch Pricing]().

The pricing calculator can help you estimate approximate monthly costs for using this solution.

To use the pricing calculator to estimate your monthly solution costs

1. Open the [Amazon CloudWatch pricing calculator]().
2. In the **Metrics** section, for **Number of metrics**, enter **`broker_metrics_count + producer_metrics_count + consumer_metrics_count`**. Calculate these as follows:
   - `broker_metrics_count` = (33 + average number of disk paths per EC2 host) * number_of_ec2_broker_hosts
   - `producer_metrics_count` = (3 * average_number_of_topics_per_producer_host + 3) * number_of_ec2_producer_hosts

- `consumer_metrics_count` = (2 * average_number_of_topics_per_consumer_host + 3) * number_of_ec2_consumer_hosts

3. In the **APIs** section, for **Number of API requests**, enter **`43200 * number of EC2 instances configured for this solution`**.

   By default, the CloudWatch agent performs one **PutMetricData** operation each minute for each EC2 host.

4. In the **Dashboards and Alarms** section, for **Number of Dashboards**, enter **1**.
5. You can see your monthly estimated costs at the bottom of the pricing calculator.

CloudWatch agent configuration for this solution

The CloudWatch agent is software that runs continuously and autonomously on your servers and in containerized environments. It collects metrics, logs, and traces from your infrastructure and applications and sends them to CloudWatch and X-Ray.

For more information about the CloudWatch agent, see Collect metrics, logs, and traces with the CloudWatch agent.

The agent configuration in this solution collects the foundational metrics for Kafka, JVM, and EC2. The CloudWatch agent can be configured to collect more Kafka and JVM metrics than the dashboard displays by default. For a list of all Kafka metrics that you can collect, see Collect Kafka metrics. For a list of all JVM metrics that you can collect, see Collect JVM metrics. For a list of EC2 metrics, see Metrics collected by the CloudWatch agent on Linux and macOS instances.

Expose JMX ports for the Kafka broker, producer, and consumer roles

The CloudWatch agent relies on JMX to collect the metrics related to the Kafka brokers, producers, and consumers. To make this possible, you must expose the JMX port on your servers and applications.

For Kafka brokers, you must use the `JMX_PORT` environment variable to set the port. You'll have to restart the brokers after you set this environment variable. Review the starting scripts and configuration files of your application to find the best place to add these arguments.

For example, for Linux and macOS systems, you can use the following command to set the JMX port. Be sure to specify an unused port number.

```
export JMX_PORT=port-number
```

For Kafka producers and consumers, instructions for exposing the JMX port depend on the workload type you are using for your producer or consumer JVM application. See the documentation for your application to find these instructions.

In general, to enable a JMX port for monitoring and management, you would set the following system properties for your JVM application. The following example sets up unauthenticated JMX. If your security policies/requirements require you to enable JMX with password authentication or SSL for remote access, refer to the JMX documentation to set the required property.

```
-Dcom.sun.management.jmxremote
-Dcom.sun.management.jmxremote.port=port-number
-Dcom.sun.management.jmxremote.authenticate=false
-Dcom.sun.management.jmxremote.ssl=false
```

To verify the JMX port, run `ps aux | grep jmxremote.port`. The results should show that the JMX port was set on the JVM processes.

Agent configuration for this solution

The metrics collected by the agent are defined in the agent configuration. The solution provides agent configurations to collect the recommended metrics with suitable dimensions for the solution's dashboard. Each Kafka role, such as broker, producer, or consumer, has its own agent configuration that enables the collection of Kafka metrics and underlying JVM and EC2 metrics.

The steps for deploying the solution are described later in Deploy the agent for your solution. The following information is intended to help you understand how to customize the agent configuration for your environment.

You must customize some parts of the following agent configuration for your environment:

- The JMX port number is the port number that you configured in the previous section of this documentation. The port number is in the `endpoint` line in the configuration.
- `ClusterName`– This is used as a dimension for broker metrics collected. Provide a meaningful name that represents the cluster grouping for the instances that run the Kafka broker.
- `ProcessGroupName`– This is used as a dimension for JVM metrics collected for brokers. Provide the same value as you provide for `ClusterName`. This enables viewing the JVM metrics of the same Kafka broker group as the broker metrics in the solution dashboard.

- `ProducerGroupName`– This is used as a dimension for producer metrics collected. Provide a meaningful name that represents the group of producer instances. For this value, you can specify your producer application or service that you want to use for a combined view of producer metrics in the solution dashboard.
- `ConsumerGroupName`– This is used as a dimension for consumer metrics collected. Provide a meaningful name that represents the group of consumer instances This is not the same as the consumer group concept in Kafka. This is just a grouping dimension where you can specify your consumer application or service that you want to use for a combined view of consumer metrics in the solution dashboard

For example, if you have two Kafka clusters running in the same account, one for the `order-processing` application and another for the `inventory-management` application, you should set the `ClusterName` and `ProcessGroupName` dimensions accordingly in the agent configuration of the broker instance.

- For the `order-processing` cluster broker instances, set `ClusterName=order-processing` and `ProcessGroupName=order-processing`.
- For the `inventory-management` cluster broker instances, set `ClusterName=inventory-management` and `ProcessGroupName=inventory-management`.
- Similarly, set the `ProducerGroupName` for producer instances and `ConsumerGroupName` for consumer instances based on their respective applications.

When you correctly set the above dimensions, the solution dashboard will automatically group the metrics based on the `ClusterName`, `ProducerGroupName`, and `ConsumerGroupName` dimensions. The dashboard will include dropdown options to select and view metrics for specific clusters and groups, allowing you to monitor the performance of individual clusters and groups separately.

Be sure to deploy the relevant agent configuration to the correct EC2 instances. Each configuration will be stored as a separate Parameter in SSM's Parameter Store, as detailed later in [Step 2: Store the recommended CloudWatch agent configuration file in Systems Manager Parameter Store](#).

The following instructions describe the situation where the producer, consumer, and broker roles are deployed to separate EC2 instances, without any overlap. If you are running multiple Kafka roles on the same EC2 instances, see [Configure the agent for multiple Kafka roles on the same instance](#) for more information.

Agent configuration for Kafka broker agents

Use the following CloudWatch agent configuration on EC2 instances where Kafka broker agents are deployed. Replace *ClusterName* with the name of the cluster to use to group these metrics for a unified view. The value you specify for *ClusterName* is used as both the `ClusterName` dimension and the `ProcessGroupName` dimension. Replace *port-number* with the JMX port of your Kafka server. If JMX was enabled with password authentication or SSL for remote access, see Collect Java Management Extensions (JMX) metrics for information about setting up TLS or authorization as required.

The EC2 metrics shown in this configuration (configuration shown outside the JMX block) only work for Linux and macOS instances. If you are using Windows instances, you can choose to omit these metrics in the configuration. For information about metrics collected on Windows instances, see Metrics collected by the CloudWatch agent on Windows Server instances.

```
{
  "metrics": {
    "namespace": "CWAgent",
    "append_dimensions": {
      "InstanceId": "${aws:InstanceId}"
    },
    "metrics_collected": {
      "jmx": [
        {
          "endpoint": "localhost:port-number",
          "kafka": {
            "measurement": [
              "kafka.request.time.avg",
              "kafka.request.failed",
              "kafka.request.count",
              "kafka.purgatory.size",
              "kafka.partition.under_replicated",
              "kafka.partition.offline",
              "kafka.network.io",
              "kafka.leader.election.rate",
              "kafka.isr.operation.count"
            ]
          },
          "append_dimensions": {
            "ClusterName": "ClusterName"
          }
        },
```

```
        {
          "endpoint": "localhost:port-number",
          "jvm": {
            "measurement": [
              "jvm.classes.loaded",
              "jvm.gc.collections.count",
              "jvm.gc.collections.elapsed",
              "jvm.memory.heap.committed",
              "jvm.memory.heap.max",
              "jvm.memory.heap.used",
              "jvm.memory.nonheap.committed",
              "jvm.memory.nonheap.max",
              "jvm.memory.nonheap.used",
              "jvm.threads.count"
            ]
          },
          "append_dimensions": {
            "ProcessGroupName": "ClusterName"
          }
        }
      ],
      "disk": {
        "measurement": [
          "used_percent"
        ]
      },
      "mem": {
        "measurement": [
          "used_percent"
        ]
      },
      "swap": {
        "measurement": [
          "used_percent"
        ]
      },
      "netstat": {
        "measurement": [
          "tcp_established",
          "tcp_time_wait"
        ]
      }
    }
  }
```

```
}
```

Agent configuration for Kafka producers

Use the following CloudWatch agent configuration on Amazon EC2 instances where Kafka producers are deployed. Replace *ProducerGroupName* with the name of the application or group that you want to use to group your metrics for a unified view. Replace *port-number* with the JMX port of your Kafka producer application.

The solution doesn't enable JVM metrics for Kafka producers because the solution dashboard doesn't display JVM metrics related to JVM for producers. You can customize the agent configuration to emit JVM metrics as well, however, JVM metrics related to producers are not visible on the solution dashboard.

```
{
  "metrics": {
    "namespace": "CWAgent",
    "append_dimensions": {
      "InstanceId": "${aws:InstanceId}"
    },
    "metrics_collected": {
      "jmx": [
        {
          "endpoint": "localhost:port-number",
          "kafka-producer": {
            "measurement": [
              "kafka.producer.request-rate",
              "kafka.producer.byte-rate",
              "kafka.producer.request-latency-avg",
              "kafka.producer.response-rate",
              "kafka.producer.record-error-rate",
              "kafka.producer.record-send-rate"
            ]
          },
          "append_dimensions": {
            "ProducerGroupName": "ProducerGroupName"
          }
        }
      ]
    }
  }
}
```

Agent configuration for Kafka consumers

Use the following CloudWatch agent configuration on EC2 instances where Kafka consumers are running. Replace *ConsumerGroupName* with the name of the application or group to use to group these metrics for a unified view. Replace *port-number* with the JMX port of your Kafka consumer application.

The solution doesn't enable JVM metrics for Kafka consumers because the solution dashboard doesn't display JVM metrics related to JVM for consumers. You can customize the agent configuration to emit JVM metrics as well, however JVM metrics related to consumer are not visible on the solution dashboard.

```
{
  "metrics": {
    "append_dimensions": {
      "InstanceId": "${aws:InstanceId}"
    },
    "metrics_collected": {
      "jmx": [
        {
          "endpoint": "localhost:port-number",
          "kafka-consumer": {
            "measurement": [
              "kafka.consumer.fetch-rate",
              "kafka.consumer.total.bytes-consumed-rate",
              "kafka.consumer.records-consumed-rate",
              "kafka.consumer.bytes-consumed-rate",
              "kafka.consumer.records-lag-max"
            ]
          },
          "append_dimensions": {
            "ConsumerGroupName": "ConsumerGroupName"
          }
        }
      ]
    }
  }
}
```

Deploy the agent for your solution

There are several approaches for installing the CloudWatch agent, depending on the use case. We recommend using Systems Manager for this solution. It provides a console experience and makes it simpler to manage a fleet of managed servers within a single Amazon account. The instructions in this section use Systems Manager and are intended for when you don't have the CloudWatch agent running with existing configurations. You can check whether the CloudWatch agent is running by following the steps in Verify that the CloudWatch agent is running.

If you are already running the CloudWatch agent on the EC2 hosts where the workload is deployed and managing the agent configurations, you can skip the instructions in this section and follow your existing deployment mechanism to update the configuration. Be sure to merge the agent configuration according to the role (broker, producer, or consumer) with your existing agent configuration, and then deploy the merged configuration. If you are using Systems Manager to store and manage the configuration for the CloudWatch agent, you can merge the configuration to the existing parameter value. For more information, see Managing CloudWatch agent configuration files.

> **Note**
>
> Using Systems Manager to deploy the following CloudWatch agent configurations will replace or overwrite any existing CloudWatch agent configuration on your EC2 instances. You can modify this configuration to suit your unique environment or use case. The metrics defined in this solution are the minimum required for the recommended dashboard.

The deployment process includes the following steps:

- Step 1: Ensure that the target EC2 instances have the required IAM permissions.
- Step 2: Store the recommended agent configuration file in the Systems Manager Parameter Store.
- Step 3: Install the CloudWatch agent on one or more EC2 instances using an Amazon CloudFormation stack.
- Step 4: Verify the agent setup is configured properly.

You must repeat these steps based on whether your broker, producer, and consumer are deployed on the same EC2 instance or different instances. For example, if the Kafka broker, producer, and consumers are getting deployed on separate instances without overlap, you must repeat these steps three times with the appropriate agent configurations for broker, producer, and consumer EC2 instances.

Step 1: Ensure the target EC2 instances have the required IAM permissions

You must grant permission for Systems Manager to install and configure the CloudWatch agent. You must also grant permission for the CloudWatch agent to publish telemetry from your EC2 instance to CloudWatch. Make sure that the IAM role attached to the instance has the **CloudWatchAgentServerPolicy** and **AmazonSSMManagedInstanceCore** IAM policies attached.

- To create a role, see Create IAM roles to use with the CloudWatch agent on Amazon EC2 instances.
- After the role is created, attach the role to your EC2 instances. Follow the steps in Launch an instance with an IAM role to attach a role while launching a new EC2 instance. To attach a role to an existing EC2 instance, follow the steps in Attach an IAM role to an instance.

Step 2: Store the recommended CloudWatch agent configuration file in Systems Manager Parameter Store

Parameter Store simplifies the installation of the CloudWatch agent on an EC2 instance by securely storing and managing configuration parameters, eliminating the need for hard-coded values. This ensures a more secure and flexible deployment process, enabling centralized management and easier updates to configurations across multiple instances.

Use the following steps to store the recommended CloudWatch agent configuration file as a parameter in Parameter Store.

To create the CloudWatch agent configuration file as a parameter

1. Open the Amazon Systems Manager console at https://console.amazonaws.cn/systems-manager/.
2. From the navigation pane, choose **Application Management**, **Parameter Store**.
3. Follow these steps to create a new parameter for the configuration.

    a. Choose **Create parameter**.
    b. Provide a name for the parameter that will store your CloudWatch agent configuration, such as **`AmazonCloudWatch-Kafka-Producer-Configuration`** for producers,

AmazonCloudWatch-Kafka-Consumer-Configuration for consumers, or **AmazonCloudWatch-Kafka-Broker-Configuration** for brokers. If you have multiple Kafka roles on a single EC2, name the roles accordingly for easier identification. This value will later be used to distribute this configuration to the agent running on your EC2 instance.

c. For **Parameter tier**, choose **Standard**.

d. For **Type**, choose **String**.

e. For **Data type**, choose **text**.

f. In the **Value** box, paste the full text of the CloudWatch agent configuration. Be sure to select the JSON block for the Kafka role that this instance is hosting. Refer to the configuration provided in Agent configuration for Kafka broker agents, Agent configuration for Kafka producers, and Agent configuration for Kafka consumers when storing the configuration for broker, producer, and consumer respectively. If you are running multiple Kafka roles on the same EC2 instance, be sure to merge the configuration if required as described in Configure the agent for multiple Kafka roles on the same instance on the same instance

g. Choose **Create parameter**.

Step 3: Install the CloudWatch agent and apply the configuration using an Amazon CloudFormation template

You can use Amazon CloudFormation to install the agent and configure it to use the CloudWatch agent configuration that you created in the previous steps.

To install and configure the CloudWatch agent for this solution

1. Open the Amazon CloudFormation **Quick create stack** wizard using this link: https://console.amazonaws.cn/cloudformation/home?#/stacks/quickcreate?templateURL=https://aws-observability-solutions-prod-us-east-1.s3.us-east-1.amazonaws.com/CloudWatchAgent/CFN/v1.0.0/cw-agent-installation-template-1.0.0.json.
2. Verify that the selected Region on the console is the Region where the Kafka workload is running.
3. For **Stack name**, enter a name to identity this stack, such as **CWAgentInstallationStack**.
4. In the **Parameters** section, specify the following:

a. For **CloudWatchAgentConfigSSM**, enter the name of the Systems Manager parameter for the agent configuration that you created earlier, such as **`AmazonCloudWatch-Kafka-Broker-Configuration`** for brokers, **`AmazonCloudWatch-Kafka-Producer-Configuration`** for producers, and **`AmazonCloudWatch-Kafka-Consumer-Configuration`** for consumers.

b. To select the target instances, you have two options.

   i. For **InstanceIds**, specify a comma-delimited list of instance IDs list of instance IDs where you want to install the CloudWatch agent with this configuration. You can list a single instance or several instances.

   ii. If you are deploying at scale, you can specify the **TagKey** and the corresponding **TagValue** to target all EC2 instances with this tag and value. If you specify a **TagKey**, you must specify a corresponding **TagValue**. (For an Auto Scaling group, specify **`aws:autoscaling:groupName`** for the **TagKey** and specify the Auto Scaling group name for the **TagValue** to deploy to all instances within the Auto Scaling group.)

   If you specify both the **InstanceIds** and the **TagKeys** parameters, the **InstanceIds** will take precedence and the tags will be ignored.

5. Review the settings, then choose **Create stack**.

If you want to edit the template file first to customize it, choose the **Upload a template file** option under **Create Stack Wizard** to upload the edited template. For more information, see Creating a stack on Amazon CloudFormation console. You can use the following link to download the template: https://aws-observability-solutions-prod-us-east-1.s3.us-east-1.amazonaws.com/CloudWatchAgent/CFN/v1.0.0/cw-agent-installation-template-1.0.0.json.

> **Note**
>
> After this step is completed, this Systems Manager parameter will be associated with the CloudWatch agents running in the targeted instances. This means that:
>
> 1. If the Systems Manager parameter is deleted, the agent will stop.
> 2. If the Systems Manager parameter is edited, the configuration changes will automatically apply to the agent at the scheduled frequency which is 30 days by default.

3. If you want to immediately apply changes to this Systems Manager parameter, you must run this step again. For more information about associations, see Working with associations in Systems Manager.

Step 4: Verify the agent setup is configured properly

You can verify whether the CloudWatch agent is installed by following the steps in Verify that the CloudWatch agent is running. If the CloudWatch agent is not installed and running, make sure you have set up everything correctly.

- Be sure you have attached a role with correct permissions for the EC2 instance as described in Step 1: Ensure the target EC2 instances have the required IAM permissions.
- Be sure you have correctly configured the JSON for the Systems Manager parameter. Follow the steps in ???.

If everything is set up correctly, then you should see the Kafka metrics being published to CloudWatch. You can check the CloudWatch console to verify they are being published.

To verify that Kafka metrics are being published to CloudWatch

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. Choose **Metrics**, **All metrics**.
3. Make sure you've selected the Region where you deployed the solution, and choose **Custom namespaces**, **CWAgent**.
4. Search for the metrics mentioned in the agent configuration section of this document, such as `kafka.partition.offline` for brokers, `kafka.consumer.fetch.rate` for consumers, or `kafka.producer.request-rate` for producers. If you see results for these metrics, then the metrics are being published to CloudWatch.

Create the Kafka solution dashboard

This dashboard displays the newly emitted metrics for both Kafka and the underlying JVM. This dashboard provides a top contributor view for the health of your Kafka workload, across producers, brokers, and consumers. The top contributor view displays the top 10 per metric widget. This allows you to identify outliers at a glance.

The solution dashboard doesn't display EC2 metrics. To view EC2 metrics, you'll need to use the EC2 automatic dashboard to see EC2 vended metrics and use the EC2 console dashboard to see EC2 metrics that are collected by the CloudWatch agent. For more information about automatic dashboards for Amazon services, see Viewing a CloudWatch dashboard for a single Amazon service.

To create the dashboard, you can use the following options:

- Use CloudWatch console to create the dashboard.
- Use Amazon CloudFormation console to deploy the dashboard.
- Download the Amazon CloudFormation infrastructure as code and integrate it as part of your continuous integration (CI) automation.

By using the CloudWatch console to create a dashboard, you can preview the dashboard before actually creating and being charged.

> **Note**
>
> The dashboard created with Amazon CloudFormation in this solution displays metrics from the Region where the solution is deployed. Be sure to create the Amazon CloudFormation stack in the Region where your JVM and Kafka metrics are published.
> If you've specified a custom namespace other than `CWAgent` in the CloudWatch agent configuration, you'll have to change the Amazon CloudFormation template for the dashboard to replace `CWAgent` with the customized namespace you are using.

To create the dashboard via CloudWatch Console

> **Note**
>
> Solution dashboards currently display garbage collection-related metrics only for the G1 Garbage Collector, which is the default collector for the latest Java versions. If you are using a different garbage collection algorithm, the widgets pertaining to garbage collection are empty. However, you can customize these widgets by changing the dashboard CloudFormation template and applying the appropriate garbage collection type to the name dimension of the garbage collection-related metrics. For example, if you are using parallel garbage collection, change the **`name=\"G1 Young`**

Generation\" to **name=\"Parallel GC\"** of the garbage collection count metric jvm.gc.collections.count.

1. Open the CloudWatch Console **Create Dashboard** using this link: https://console.aws.amazon.com/cloudwatch/home?#dashboards?dashboardTemplate=ApacheKafkaOnEc2&referrer=os-catalog .
2. Verify that the selected Region on the console is the Region where the Kafka workload is running.
3. Enter the name of the dashboard, then choose **Create Dashboard**.

   To easily differentiate this dashboard from similar dashboards in other Regions, we recommend including the Region name in the dashboard name, such as **KafkaDashboard-us-east-1**.
4. Preview the dashboard and choose **Save** to create the dashboard.

To create the dashboard via Amazon CloudFormation

1. Open the Amazon CloudFormation **Quick create stack** wizard using this link: https://console.amazonaws.cn/cloudformation/home?#/stacks/quickcreate?templateURL=https://aws-observability-solutions-prod-us-east-1.s3.us-east-1.amazonaws.com/Kafka_EC2/CloudWatch/CFN/v1.0.0/dashboard-template-1.0.0.json.
2. Verify that the selected Region on the console is the Region where the Kafka workload is running.
3. For **Stack name**, enter a name to identity this stack, such as **KafkaDashboardStack**.
4. In the **Parameters** section, specify the name of the dashboard under the **DashboardName** parameter.

   To easily differentiate this dashboard from similar dashboards in other Regions, we recommend including the Region name in the dashboard name, such as **KafkaDashboard-us-east-1**.
5. Acknowledge access capabilities for transforms under **Capabilities and transforms**. Note that CloudFormation doesn't add any IAM resources.
6. Review the settings, then choose **Create stack**.

7. After the stack status is **CREATE_COMPLETE**, choose the **Resources** tab under the created stack and then choose the link under **Physical ID** to go to the dashboard. You can also access the dashboard in the CloudWatch console by choosing **Dashboards** in the left navigation pane of the console, and finding the dashboard name under **Custom Dashboards**.

If you want to edit the template file to customize it for any purpose, you can use **Upload a template file** option under **Create Stack Wizard** to upload the edited template. For more information, see [Creating a stack on Amazon CloudFormation console](). You can use this link to download the template: https://aws-observability-solutions-prod-us-east-1.s3.us-east-1.amazonaws.com/Kafka_EC2/CloudWatch/CFN/v1.0.0/dashboard-template-1.0.0.json.

> **Note**
>
> Solution dashboards currently display garbage collection-related metrics only for the G1 Garbage Collector, which is the default collector for the latest Java versions. If you are using a different garbage collection algorithm, the widgets pertaining to garbage collection are empty. However, you can customize these widgets by changing the dashboard CloudFormation template and applying the appropriate garbage collection type to the name dimension of the garbage collection-related metrics. For example, if you are using parallel garbage collection, change the **`name=\"G1 Young Generation\"`** to **`name=\"Parallel GC\"`** of the garbage collection count metric `jvm.gc.collections.count`.

Get started with the Kafka dashboard

Here are a few tasks that you can try out with the new Kafka dashboard. These tasks allow you to validate that the dashboard is working correctly and provide you some hands-on experience using it to monitor a Kafka cluster. As you try these out, you'll get familiar with navigating the dashboard and interpreting the visualized metrics.

Using the dropdown lists

The dashboard provides dropdown lists at the top that you can use to filter and select the specific Kafka cluster, producer, and consumer groups that you want to monitor.

- To display metrics for a specific Kafka cluster, select that cluster name in the **Kafka Cluster** dropdown list.

- To display metrics for a specific Kafka producer group, select that producer group name in the **Kafka Producer** dropdown list.
- To display metrics for a specific Kafka consumer group, select that consumer group name in the **Kafka Consumer Group** dropdown list.

Verify cluster health

From the **Cluster Overview** section, find the **Partitions Under Replicated** and **In-Sync Replicas** widgets. These should ideally be zero or a small number. A large value for any of these metrics could indicate issues with the Kafka cluster that need investigation.

Investigate broker performance

In the **Brokers** section, find the **Failed Fetch Requests** and **Failed Producer Requests** widgets. These show the number of failed requests for fetch and produce operations, respectively. High failure rates could indicate issues with the brokers or network connectivity that require further investigation.

Monitor producer performance

In the **Producer Group Overview** section, find the **Average Request Rate**, **Average Request Latency**, and **Average Record Send/Error Rate** widgets. These will give you an overview of how the producers in the selected group are performing. You can also drill down to view metrics for specific producers and topics in the **Producers** section.

Monitor consumer lag

In the **Consumer Group Overview** section, find the **Consumer Lag** widget. This shows how far behind the consumers are in processing messages from the latest offsets in the partitions they are subscribed to. Ideally, the consumer lag should be low or zero. A high consumer lag could indicate that the consumers are unable to keep up with the rate of data production, leading to potential data loss or delays in processing. You can also drill down to view metrics for specific consumers and topics in the **Consumers** section.

Configure the agent for multiple Kafka roles on the same instance

The individual configurations for Kafka roles listed in [CloudWatch agent configuration for this solution](#) apply only when the producer, consumer, and broker roles are deployed on separate EC2 instances, without any overlap. If you are running multiple Kafka roles on the same Amazon EC2 instances, you have two options:

- Create a single agent configuration file which lists and configures all metrics for all the Kafka roles deployed on that instance. If you are going to use Systems Manager to manage agent configuration, this is the preferred option.

  If you choose this option and the multiple Kafka roles are part of the same JVM process, you must specify the same endpoint for each Kafka role in the agent configuration. If the multiple Kafka roles are part of different JVM processes, the endpoint for each role can be different depending on the JMX port set for that process.
- Create separate agent configuration files for each Kafka role, and configure the agent to apply both configuration files. For instructions for applying multiple configuration files, see Multiple CloudWatch agent configuration files.

The following example shows a CloudWatch agent configuration where the producer and consumer roles are running on one instance as part of the same JVM process. In this case, the port number must be the same in both the producer and consumer parts of the configuration below. If instead the two roles were running as part of different JVM processes, you could specify different port numbers for each, according to the JMX port of each individual JVM process.

```
{
  "metrics": {
    "namespace": "CWAgent",
    "append_dimensions": {
      "InstanceId": "${aws:InstanceId}"
    },
    "metrics_collected": {
      "jmx": [
        {
          "endpoint": "localhost:port-number",
          "kafka-producer": {
            "measurement": [
              "kafka.producer.request-rate",
              "kafka.producer.byte-rate",
              "kafka.producer.request-latency-avg",
              "kafka.producer.response-rate",
              "kafka.producer.record-error-rate",
              "kafka.producer.record-send-rate"
            ]
```

```
          },
          "append_dimensions": {
            "ProducerGroupName": "ProducerGroupName"
          }
        },
        {
          "endpoint": "localhost:port-number",
          "kafka-consumer": {
            "measurement": [
              "kafka.consumer.fetch-rate",
              "kafka.consumer.total.bytes-consumed-rate",
              "kafka.consumer.records-consumed-rate",
              "kafka.consumer.bytes-consumed-rate",
              "kafka.consumer.records-lag-max"
            ]
          },
          "append_dimensions": {
            "ConsumerGroupName": "ConsumerGroupName"
          }
        }
      ]
    }
  }
}
```

CloudWatch solution: Tomcat workload on Amazon EC2

This solution helps you configure out-of-the-box metric collection using CloudWatch agents for Tomcat server running on EC2 instances. Additionally, it helps you set up a pre-configured CloudWatch dashboard. For general information about all CloudWatch observability solutions, see CloudWatch observability solutions.

Topics

Requirements

This solution is relevant for the following conditions:

- Supported versions: Tomcat versions 9, 10.1, and 11 (beta)
- Compute: Amazon EC2
- Supports up to 500 EC2 instances across all Tomcat workloads in a given Amazon Web Services Region
- Latest version of CloudWatch agent
- SSM agent installed on EC2 instance

> **Note**
>
> Amazon Systems Manager (SSM agent) is pre-installed on some Amazon Machine Images (AMIs) provided by Amazon and trusted third-parties. If the agent isn't installed, you can install it manually using the procedure for your operating system type.
>
> - Manually installing and uninstalling SSM Agent on EC2 instances for Linux
> - Manually installing and uninstalling SSM Agent on EC2 instances for macOS
> - Manually installing and uninstalling SSM Agent on EC2 instances for Windows Server

Benefits

The solution delivers Tomcat server monitoring, providing valuable insights for the following use cases:

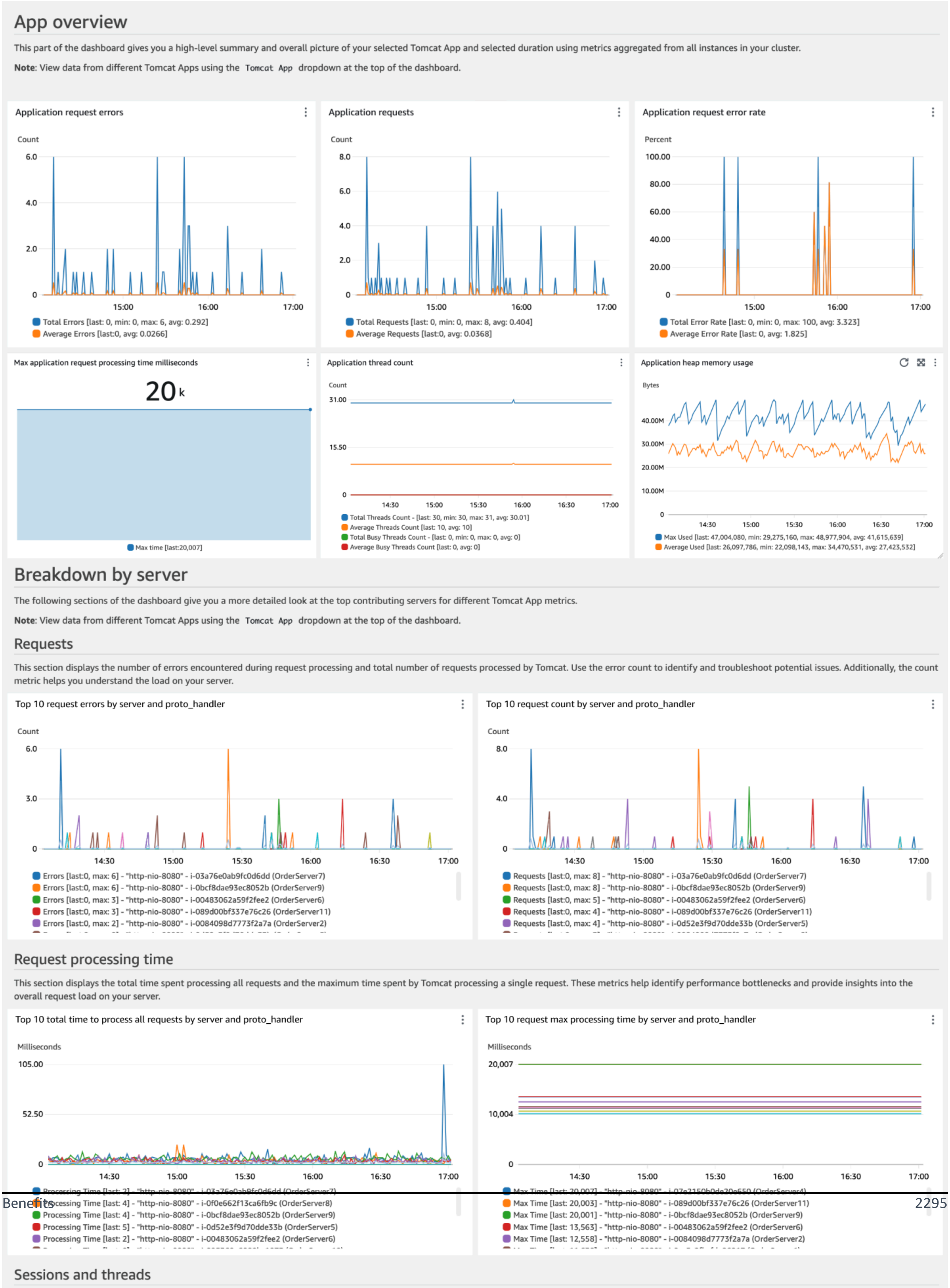
- Detect Tomcat server errors and performance issues.
- Monitor network traffic for data transfer problems.
- Track thread usage and active user sessions.
- Analyze underlying JVM performance for Tomcat server.

Below are the key advantages of the solution:

- Automates metric collection for Apache Tomcat and the underlying JVM using CloudWatch agent configuration, eliminating manual instrumentation.

- Provides a pre-configured, consolidated CloudWatch dashboard for Apache Tomcat and JVM metrics. The dashboard will automatically handle metrics from new Tomcat EC2 instances configured using the solution, even if those metrics don't exist when you first create the dashboard. It also allows you to group the metrics into logical applications for easier focus and management.

The following image is an example of the dashboard for this solution.


App overview
This part of the dashboard gives you a high-level summary and overall picture of your selected Tomcat App and selected duration using metrics aggregated from all instances in your cluster.
Note: View data from different Tomcat Apps using the Tomcat App dropdown at the top of the dashboard.
Application request errors
Total Errors [last: 0, min: 0, max: 6, avg: 0.292]
Average Errors [last:0, avg: 0.0266]
Application requests
Total Requests [last: 0, min: 0, max: 8, avg: 0.404]
Average Requests [last:0, avg: 0.0368]
Application request error rate
Total Error Rate [last: 0, min: 0, max: 100, avg: 3.323]
Average Error Rate [last: 0, avg: 1.825]
Max application request processing time milliseconds
20k
Max time [last:20,007]
Application thread count
Total Threads Count - [last: 30, min: 30, max: 31, avg: 30.01]
Average Threads Count [last: 10, avg: 10]
Total Busy Threads Count - [last: 0, min: 0, max: 0, avg: 0]
Average Busy Threads Count [last: 0, avg: 0]
Application heap memory usage
Max Used [last: 47,004,080, min: 29,275,160, max: 48,977,904, avg: 41,615,639]
Average Used [last: 26,097,786, min: 22,098,143, max: 34,470,531, avg: 27,423,532]
Breakdown by server
The following sections of the dashboard give you a more detailed look at the top contributing servers for different Tomcat App metrics.
Note: View data from different Tomcat Apps using the Tomcat App dropdown at the top of the dashboard.
Requests
This section displays the number of errors encountered during request processing and total number of requests processed by Tomcat. Use the error count to identify and troubleshoot potential issues. Additionally, the count metric helps you understand the load on your server.
Top 10 request errors by server and proto_handler
Errors [last:0, max: 6] - "http-nio-8080" - i-03a76e0ab9fc0d6dd (OrderServer7)
Errors [last:0, max: 6] - "http-nio-8080" - i-0bcf8dae93ec8052b (OrderServer9)
Errors [last:0, max: 3] - "http-nio-8080" - i-00483062a59f2fee2 (OrderServer6)
Errors [last:0, max: 3] - "http-nio-8080" - i-089d00bf337e76c26 (OrderServer11)
Errors [last:0, max: 2] - "http-nio-8080" - i-0084098d7773f2a7a (OrderServer2)
Top 10 request count by server and proto_handler
Requests [last:0, max: 8] - "http-nio-8080" - i-03a76e0ab9fc0d6dd (OrderServer7)
Requests [last:0, max: 8] - "http-nio-8080" - i-0bcf8dae93ec8052b (OrderServer9)
Requests [last:0, max: 5] - "http-nio-8080" - i-00483062a59f2fee2 (OrderServer6)
Requests [last:0, max: 4] - "http-nio-8080" - i-089d00bf337e76c26 (OrderServer11)
Requests [last:0, max: 4] - "http-nio-8080" - i-0d52e3f9d70dde33b (OrderServer5)
Request processing time
This section displays the total time spent processing all requests and the maximum time spent by Tomcat processing a single request. These metrics help identify performance bottlenecks and provide insights into the overall request load on your server.
Top 10 total time to process all requests by server and proto_handler
Processing Time [last: 4] - "http-nio-8080" - i-0f0e662f13ca6fb9c (OrderServer8)
Processing Time [last: 4] - "http-nio-8080" - i-0bcf8dae93ec8052b (OrderServer9)
Processing Time [last: 5] - "http-nio-8080" - i-0d52e3f9d70dde33b (OrderServer5)
Processing Time [last: 2] - "http-nio-8080" - i-00483062a59f2fee2 (OrderServer6)
Top 10 request max processing time by server and proto_handler
Max Time [last: 20,003] - "http-nio-8080" - i-089d00bf337e76c26 (OrderServer11)
Max Time [last: 20,001] - "http-nio-8080" - i-0bcf8dae93ec8052b (OrderServer9)
Max Time [last: 13,563] - "http-nio-8080" - i-00483062a59f2fee2 (OrderServer6)
Max Time [last: 12,558] - "http-nio-8080" - i-0084098d7773f2a7a (OrderServer2)
Sessions and threads

Costs

This solution creates and uses resources in your account. You are charged for standard usage, including the following:

- All metrics collected by the CloudWatch agent are charged as custom metrics. The number of metrics used by this solution depends on the number of EC2 hosts.
  - Each Tomcat host configured for the solution publishes a total of 27 metrics plus one metric (`disk_used_percent`) for which the metric count depends on number of disk paths for that host.
- One custom dashboard.
- API operations requested by the CloudWatch agent to publish the metrics. With the default configuration for this solution, the CloudWatch agent calls the **PutMetricData** once every minute. This means the **PutMetricData** API will be called `30*24*60=43,200` in a 30-day month for each EC2 host.

For more information about CloudWatch pricing, see [Amazon CloudWatch Pricing](https://aws.amazon.com/cloudwatch/pricing/).

The pricing calculator can help you estimate approximate monthly costs for using this solution.

To use the pricing calculator to estimate your monthly solution costs

1. Open the [Amazon CloudWatch pricing calculator](https://calculator.aws/#/createCalculator/CloudWatch).
2. In the **Metrics** section, for **Number of metrics**, enter **`(27 + average number of disk paths per EC2 host) * number of EC2 instances configured for this solution`**.
3. In the **APIs** section, for **Number of API requests**, enter **`43200 * number of EC2 instances configured for this solution`**.

   By default, the solution performs one **PutMetricData** operation each minute for each EC2 host.
4. In the **Dashboards and Alarms** section, for **Number of Dashboards**, enter **1**.
5. You can see your monthly estimated costs at the bottom of the pricing calculator.

CloudWatch agent configuration for this solution

The CloudWatch agent is software that runs continuously and autonomously on your servers and in containerized environments. It collects metrics, logs, and traces from your infrastructure and applications and sends them to CloudWatch and X-Ray.

For more information about the CloudWatch agent, see Collect metrics, logs, and traces with the CloudWatch agent.

The agent configuration in this solution collects the foundational metrics for Tomcat, JVM, and EC2. The CloudWatch agent can be configured to collect more JVM metrics than the dashboard displays by default. For a list of all Tomcat metrics that you can collect, see Collect Tomcat metrics. For a list of all JVM metrics that you can collect, see Collect JVM metrics. For a list of Amazon EC2 metrics, see Metrics collected by the CloudWatch agent on Linux and macOS instances.

Expose JMX ports for the Tomcat server

The CloudWatch agent relies on JMX to collect the metrics related to the Tomcat server and JVM process. To make this possible, you must expose the JMX port from your servers. To enable a JMX port for monitoring and management, you would set system properties for your Tomcat servers. You can use the environment variable `CATALINA_OPTS` to set the required system properties for Tomcat. Review the startup scripts and configuration files of your Tomcat server on the best place to set the environment variable. Be sure that you specify an unused port number. You will need to restart the server after the change.

```
export CATALINA_OPTS="-Dcom.sun.management.jmxremote -
Dcom.sun.management.jmxremote.port=<<port-number>> -
Dcom.sun.management.jmxremote.ssl=false -
Dcom.sun.management.jmxremote.authenticate=false"
```

This example sets up unauthenticated JMX. If your security policies/requirements require you to enable JMX with password authentication or SSL for remote access, refer to the JMX documentation to set the required property.

To verify the JMX port, run `ps aux | grep jmxremote.port`. The results should show that the JMX port was set on the JVM processes.

Agent configuration for Tomcat solution

The metrics collected by the agent are defined in the agent configuration. The solution provides agent configurations to collect the recommended metrics with suitable dimensions for the solution's dashboard.

The steps for deploying the solution are described later in Deploy the agent for your solution. The following information is intended to help you understand how to customize the agent configuration for your environment.

You must customize some parts of the following agent configuration for your environment:

- The JMX port number is the port number that you configured in the previous section of this documentation. The port number is in the `endpoint` line in the configuration.
- `AppName` – This is used as a dimension for the Tomcat application metrics collected. Provide a meaningful name that represents the grouping for the instances that run the Tomcat application.
- `ProcessGroupName` – This is used as a dimension for JVM metrics collected for Tomcat hosts. Provide the value which is the same as `AppName` above. This is to enable viewing the JVM metrics of the same Tomcat app group as server metrics under the solution dashboard.

For example, if you have two Tomcat apps running in the same Amazon Web Services account, one for the `billing-system` application and another for the `order-system` application, you can set the `AppName` and `ProcessGroupName` dimensions accordingly in the agent configuration of each instance.

- For the `billing-system` application instances, set `AppName=billing-system` and `ProcessGroupName=billing-system`.
- For the `order-system` application instances, set `AppName=order-system` and `ProcessGroupName=order-system`.

When you follow these guidelines, the solution will automatically group the metrics based on the `AppName` and `ProcessGroupName` dimensions. The dashboard will include dropdown options to select and view metrics for a specific Tomcat application, allowing you to monitor the performance of individual applications separately.

Agent configuration for Tomcat hosts

Use the following CloudWatch agent configuration on EC2 instances where your Tomcat applications are deployed. Configuration will be stored as a parameter in SSM's Parameter Store, as detailed later in Step 2: Store the recommended CloudWatch agent configuration file in Systems Manager Parameter Store.

Replace *AppName* with a meaningful name that represents the Tomcat application the instances are part of. Replace *port-number* with the JMX port of your Tomcat server. If JMX was enabled with password authentication or SSL for remote access, see Collect Java Management Extensions (JMX) metrics for information about setting up TLS or authorization in agent configuration as required.

The EC2 metrics shown in this configuration (configuration shown outside the JMX block) only work for Linux and macOS instances. If you are using Windows instances, you can choose to omit these metrics in the configuration. For information about metrics collected on Windows instances, see Metrics collected by the CloudWatch agent on Windows Server instances.

```
{
  "metrics": {
    "namespace": "CWAgent",
    "append_dimensions": {
      "InstanceId": "${aws:InstanceId}"
    },
    "metrics_collected": {
      "jmx": [
        {
          "endpoint": "localhost:port-number",
          "tomcat": {
            "measurement": [
              "tomcat.sessions",
              "tomcat.errors",
              "tomcat.processing_time",
              "tomcat.traffic",
              "tomcat.max_time",
              "tomcat.request_count",
              "tomcat.threads"
            ]
          },
          "append_dimensions": {
            "AppName": "AppName"
          }
        },
```

```
        {
          "endpoint": "localhost:port-number",
          "jvm": {
            "measurement": [
              "jvm.classes.loaded",
              "jvm.gc.collections.count",
              "jvm.gc.collections.elapsed",
              "jvm.memory.heap.committed",
              "jvm.memory.heap.max",
              "jvm.memory.heap.used",
              "jvm.memory.nonheap.committed",
              "jvm.memory.nonheap.max",
              "jvm.memory.nonheap.used",
              "jvm.threads.count"
            ]
          },
          "append_dimensions": {
            "ProcessGroupName": "AppName"
          }
        }
      ],
      "disk": {
        "measurement": [
          "used_percent"
        ]
      },
      "mem": {
        "measurement": [
          "used_percent"
        ]
      },
      "swap": {
        "measurement": [
          "used_percent"
        ]
      },
      "netstat": {
        "measurement": [
          "tcp_established",
          "tcp_time_wait"
        ]
      }
    }
  }
```

```
}
```

Deploy the agent for your solution

There are several approaches for installing the CloudWatch agent, depending on the use case. We recommend using Systems Manager for this solution. It provides a console experience and makes it simpler to manage a fleet of managed servers within a single Amazon account. The instructions in this section use Systems Manager and are intended for when you don't have the CloudWatch agent running with existing configurations. You can check whether the CloudWatch agent is running by following the steps in Verify that the CloudWatch agent is running.

If you are already running the CloudWatch agent on the EC2 hosts where the JVM application is deployed and managing the agent configurations, you can skip the instructions in this section and follow your existing deployment mechanism to update the configuration. Be sure to merge the agent configuration of JVM with your existing agent configuration, and then deploy the merged configuration. If you are using Systems Manager to store and manage the configuration for the CloudWatch agent, you can merge the configuration to the existing parameter value. For more information, see Managing CloudWatch agent configuration files.

> **Note**
>
> Using Systems Manager to deploy the following CloudWatch agent configurations will replace or overwrite any existing CloudWatch agent configuration on your EC2 instances. You can modify this configuration to suit your unique environment or use case. The metrics defined in this solution are the minimum required for the recommended dashboard.

The deployment process includes the following steps:

- Step 1: Ensure that the target EC2 instances have the required IAM permissions.
- Step 2: Store the recommended agent configuration file in the Systems Manager Parameter Store.
- Step 3: Install the CloudWatch agent on one or more EC2 instances using an Amazon CloudFormation stack.
- Step 4: Verify the agent setup is configured properly.

Step 1: Ensure the target EC2 instances have the required IAM permissions

You must grant permission for Systems Manager to install and configure the CloudWatch agent. You must also grant permission for the CloudWatch agent to publish telemetry from your EC2 instance to CloudWatch. Make sure that the IAM role attached to the instance has the **CloudWatchAgentServerPolicy** and **AmazonSSMManagedInstanceCore** IAM policies attached.

- To create a role, see Create IAM roles to use with the CloudWatch agent on Amazon EC2 instances.
- After the role is created, attach the role to your EC2 instances. Follow the steps in Launch an instance with an IAM role to attach a role while launching a new EC2 instance. To attach a role to an existing EC2 instance, follow the steps in Attach an IAM role to an instance.

Step 2: Store the recommended CloudWatch agent configuration file in Systems Manager Parameter Store

Parameter Store simplifies the installation of the CloudWatch agent on an EC2 instance by securely storing and managing configuration parameters, eliminating the need for hard-coded values. This ensures a more secure and flexible deployment process, enabling centralized management and easier updates to configurations across multiple instances.

Use the following steps to store the recommended CloudWatch agent configuration file as a parameter in Parameter Store.

To create the CloudWatch agent configuration file as a parameter

1. Open the Amazon Systems Manager console at https://console.amazonaws.cn/systems-manager/.
2. From the navigation pane, choose **Application Management**, **Parameter Store**.
3. Follow these steps to create a new parameter for the configuration.

    a. Choose **Create parameter**.

    b. In the **Name** box, enter a name that you'll use to reference the CloudWatch agent configuration file in later steps. For example, **`AmazonCloudWatch-Tomcat-Configuration`**.

    c. (Optional) In the **Description** box, type a description for the parameter.

    d. For **Parameter tier**, choose **Standard**.

e. For **Type**, choose **String**.

f. For **Data type**, choose **text**.

g. In the **Value** box, paste the corresponding JSON block that was listed in [Agent configuration for Tomcat hosts](). Be sure to customize the grouping dimension value and port number as described.

h. Choose **Create parameter**.

Step 3: Install the CloudWatch agent and apply the configuration using an Amazon CloudFormation template

You can use Amazon CloudFormation to install the agent and configure it to use the CloudWatch agent configuration that you created in the previous steps.

To install and configure the CloudWatch agent for this solution

1. Open the Amazon CloudFormation **Quick create stack** wizard using this link: https://console.amazonaws.cn/cloudformation/home?#/stacks/quickcreate?templateURL=https://aws-observability-solutions-prod-us-east-1.s3.us-east-1.amazonaws.com/CloudWatchAgent/CFN/v1.0.0/cw-agent-installation-template-1.0.0.json.
2. Verify that the selected Region on the console is the Region where the Tomcat workload is running.
3. For **Stack name**, enter a name to identity this stack, such as **`CWAgentInstallationStack`**.
4. In the **Parameters** section, specify the following:

    a. For **CloudWatchAgentConfigSSM**, enter the name of the Systems Manager parameter for the agent configuration that you created earlier, such as **`AmazonCloudWatch-Tomcat-Configuration`**.

    b. To select the target instances, you have two options.

        i. For **InstanceIds**, specify a comma-delimited list of instance IDs list of instance IDs where you want to install the CloudWatch agent with this configuration. You can list a single instance or several instances.

        ii. If you are deploying at scale, you can specify the **TagKey** and the corresponding **TagValue** to target all EC2 instances with this tag and value. If you specify a **TagKey**, you must specify a corresponding **TagValue**. (For an Auto Scaling group, specify

aws:autoscaling:groupName for the **TagKey** and specify the Auto Scaling group name for the **TagValue** to deploy to all instances within the Auto Scaling group.)

If you specify both the **InstanceIds** and the **TagKeys** parameters, the **InstanceIds** will take precedence and the tags will be ignored.

5. Review the settings, then choose **Create stack**.

If you want to edit the template file first to customize it, choose the **Upload a template file** option under **Create Stack Wizard** to upload the edited template. For more information, see Creating a stack on Amazon CloudFormation console. You can use the following link to download the template: https://aws-observability-solutions-prod-us-east-1.s3.us-east-1.amazonaws.com/CloudWatchAgent/CFN/v1.0.0/cw-agent-installation-template-1.0.0.json.

Note

After this step is completed, this Systems Manager parameter will be associated with the CloudWatch agents running in the targeted instances. This means that:

1. If the Systems Manager parameter is deleted, the agent will stop.
2. If the Systems Manager parameter is edited, the configuration changes will automatically apply to the agent at the scheduled frequency which is 30 days by default.
3. If you want to immediately apply changes to this Systems Manager parameter, you must run this step again. For more information about associations, see Working with associations in Systems Manager.

Step 4: Verify the agent setup is configured properly

You can verify whether the CloudWatch agent is installed by following the steps in Verify that the CloudWatch agent is running. If the CloudWatch agent is not installed and running, make sure you have set up everything correctly.

- Be sure you have attached a role with correct permissions for the EC2 instance as described in Step 1: Ensure the target EC2 instances have the required IAM permissions.
- Be sure you have correctly configured the JSON for the Systems Manager parameter. Follow the steps in ???.

If everything is set up correctly, then you should see the Tomcat metrics being published to CloudWatch. You can check the CloudWatch console to verify they are being published.

To verify that Tomcat metrics are being published to CloudWatch

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. Choose **Metrics**, **All metrics**.
3. Make sure you've selected the Region where you deployed the solution, and choose **Custom namespaces**, **CWAgent**.
4. Search for the metrics mentioned in the agent configuration section of this document, such as `tomcat.errors`. If you see results for these metrics, then the metrics are being published to CloudWatch.

Create the Tomcat solution dashboard

This dashboard displays the newly emitted metrics, showing both the Tomcat application server and the underlying JVM. This dashboard provides a top contributor view for the health of your Tomcat workload. The top contributor view displays the top 10 per metric widget. This allows you to identify outliers at a glance. The dashboard also includes an overview of the cluster by aggregating and presenting metrics across all instances, providing a high-level summary of the overall health and operational state of the cluster.

The solution dashboard doesn't display EC2 metrics. To view EC2 metrics, you'll need to use the EC2 automatic dashboard to see EC2 vended metrics and use the EC2 console dashboard to see EC2 metrics that are collected by the CloudWatch agent. For more information about automatic dashboards for Amazon Web Services services, see Viewing a CloudWatch dashboard for a single Amazon service.

To create the dashboard, you can use the following options:

- Use CloudWatch console to create the dashboard.
- Use Amazon CloudFormation console to deploy the dashboard.
- Download the Amazon CloudFormation infrastructure as code and integrate it as part of your continuous integration (CI) automation.

By using the CloudWatch console to create a dashboard, you can preview the dashboard before actually creating and being charged.

> **Note**
>
> The dashboard created with Amazon CloudFormation in this solution displays metrics from the Region where the solution is deployed. Be sure to create the Amazon CloudFormation stack in the Region where your Tomcat metrics are published.
>
> If you've specified a custom namespace other than `CWAgent` in the CloudWatch agent configuration, you'll have to change the Amazon CloudFormation template for the dashboard to replace `CWAgent` with the customized namespace you are using.

To create the dashboard via CloudWatch Console

> **Note**
>
> Solution dashboards currently display garbage collection-related metrics only for the G1 Garbage Collector, which is the default collector for the latest Java versions. If you are using a different garbage collection algorithm, the widgets pertaining to garbage collection are empty. However, you can customize these widgets by changing the dashboard CloudFormation template and applying the appropriate garbage collection type to the name dimension of the garbage collection-related metrics. For example, if you are using parallel garbage collection, change the **`name=\"G1 Young Generation\"`** to **`name=\"Parallel GC\"`** of the garbage collection count metric `jvm.gc.collections.count`.

1. Open the CloudWatch Console **Create Dashboard** using this link: https://console.aws.amazon.com/cloudwatch/home?#dashboards?dashboardTemplate=ApacheTomcatOnEc2&referrer=os-catalog .
2. Verify that the selected Region on the console is the Region where the Tomcat workload is running.
3. Enter the name of the dashboard, then choose **Create Dashboard**.

   To easily differentiate this dashboard from similar dashboards in other Regions, we recommend including the Region name in the dashboard name, such as **`TomcatDashboard-us-east-1`**.
4. Preview the dashboard and choose **Save** to create the dashboard.

To create the dashboard via Amazon CloudFormation

1. Open the Amazon CloudFormation **Quick create stack** wizard using this link: https://console.amazonaws.cn/cloudformation/home?#/stacks/quickcreate?templateURL=https://aws-observability-solutions-prod-us-east-1.s3.us-east-1.amazonaws.com/Tomcat_EC2/CloudWatch/CFN/v1.0.0/dashboard-template-1.0.0.json.
2. Verify that the selected Region on the console is the Region where the Tomcat workload is running.
3. For **Stack name**, enter a name to identity this stack, such as **`TomcatDashboard-us-east-1`**.
4. In the **Parameters** section, specify the name of the dashboard under the **DashboardName** parameter.
5. To easily differentiate this dashboard from similar dashboards in other Regions, we recommend including the Region name in the dashboard name, such as **`TomcatDashboard-us-east-1`**.
6. Acknowledge access capabilities for transforms under **Capabilities and transforms**. Note that CloudFormation doesn't add any IAM resources.
7. Review the settings, then choose **Create stack**.
8. After the stack status is **CREATE_COMPLETE**, choose the **Resources** tab under the created stack and then choose the link under **Physical ID** to go to the dashboard. You can also access the dashboard in the CloudWatch console by choosing **Dashboards** in the left navigation pane of the console, and finding the dashboard name under **Custom Dashboards**.

If you want to edit the template file to customize it for any purpose, you can use **Upload a template file** option under **Create Stack Wizard** to upload the edited template. For more information, see Creating a stack on Amazon CloudFormation console. You can use this link to download the template: https://aws-observability-solutions-prod-us-east-1.s3.us-east-1.amazonaws.com/Tomcat_EC2/CloudWatch/CFN/v1.0.0/dashboard-template-1.0.0.json.

Note

Solution dashboards currently display garbage collection-related metrics only for the G1 Garbage Collector, which is the default collector for the latest Java versions. If you are using a different garbage collection algorithm, the widgets pertaining to garbage collection are empty. However, you can customize these widgets by changing the dashboard CloudFormation template and applying the appropriate garbage collection type to the name dimension of the garbage collection-related metrics. For example, if you are using parallel garbage collection, change the **`name=\"G1 Young Generation\"`** to **`name=\"Parallel GC\"`** of the garbage collection count metric `jvm.gc.collections.count`.

Get started with the Tomcat monitoring dashboard

Here are a few tasks that you can try out with the new Tomcat dashboard. These tasks allow you to validate that the dashboard is working correctly and provide you some hands-on experience using it to monitor a Tomcat application. As you try these out, you'll get familiar with navigating the dashboard and interpreting the visualized metrics.

Using the dropdown list

The dashboard provides a dropdown list at the top that you can use to filter and select the specific Tomcat application that you want to monitor. To display metrics for a specific Tomcat application, select that application name in the **Tomcat App** dropdown list.

Verify application health

From the **App Overview** section, find the **Requests**, **Errors**, and **Error Rate** widgets. These provide a high-level summary of the application's request handling performance. Look for any abnormally high error counts or rates, which could indicate issues that need investigation.

Monitor request processing

In the **Request Processing Time** section, find the **Max Time** and **Total Time to Process All Requests** widgets. These metrics help you identify potential performance bottlenecks in request processing. Look for any servers with significantly higher max processing times compared to others.

Analyze network traffic

In the **Network Traffic** section, find the **Sent Traffic** and **Received Traffic** widgets. These show the amount of data being sent and received by the application over the network. Unexpectedly high traffic levels could indicate potential issues with network saturation or inefficient data transfer.

Investigate thread usage

In the **Sessions and Threads** section, find the **Busy Threads Count**, **Threads Count**, and **Sessions** widgets. These metrics provide insights into the application's thread management and active user sessions. Look for any servers with an abnormally high number of busy threads or sessions, which could indicate potential resource constraints.

CloudWatch solution: Amazon EC2 health

This solution helps you configure out-of-the-box metric collection using CloudWatch agents for workloads running on EC2 instances. Additionally, it helps you set up a pre-configured CloudWatch dashboard.

Topics

Requirements

This solution is relevant for the following conditions:

- Compute: Amazon EC2
- Platform: Linux and macOS
- Supports up to 500 EC2 instances in a given Amazon Web Services Region
- Latest version of CloudWatch agent
- SSM agent installed on EC2 instance

> **Note**
>
> Amazon Systems Manager (SSM agent) is pre-installed on some Amazon Machine Images (AMIs) provided by Amazon and trusted third-parties. If the agent isn't installed, you can install it manually using the procedure for your operating system type.
>
> - Manually installing and uninstalling SSM Agent on EC2 instances for Linux

- [Manually installing and uninstalling SSM Agent on EC2 instances for macOS]
- [Manually installing and uninstalling SSM Agent on EC2 instances for Windows Server]

Benefits

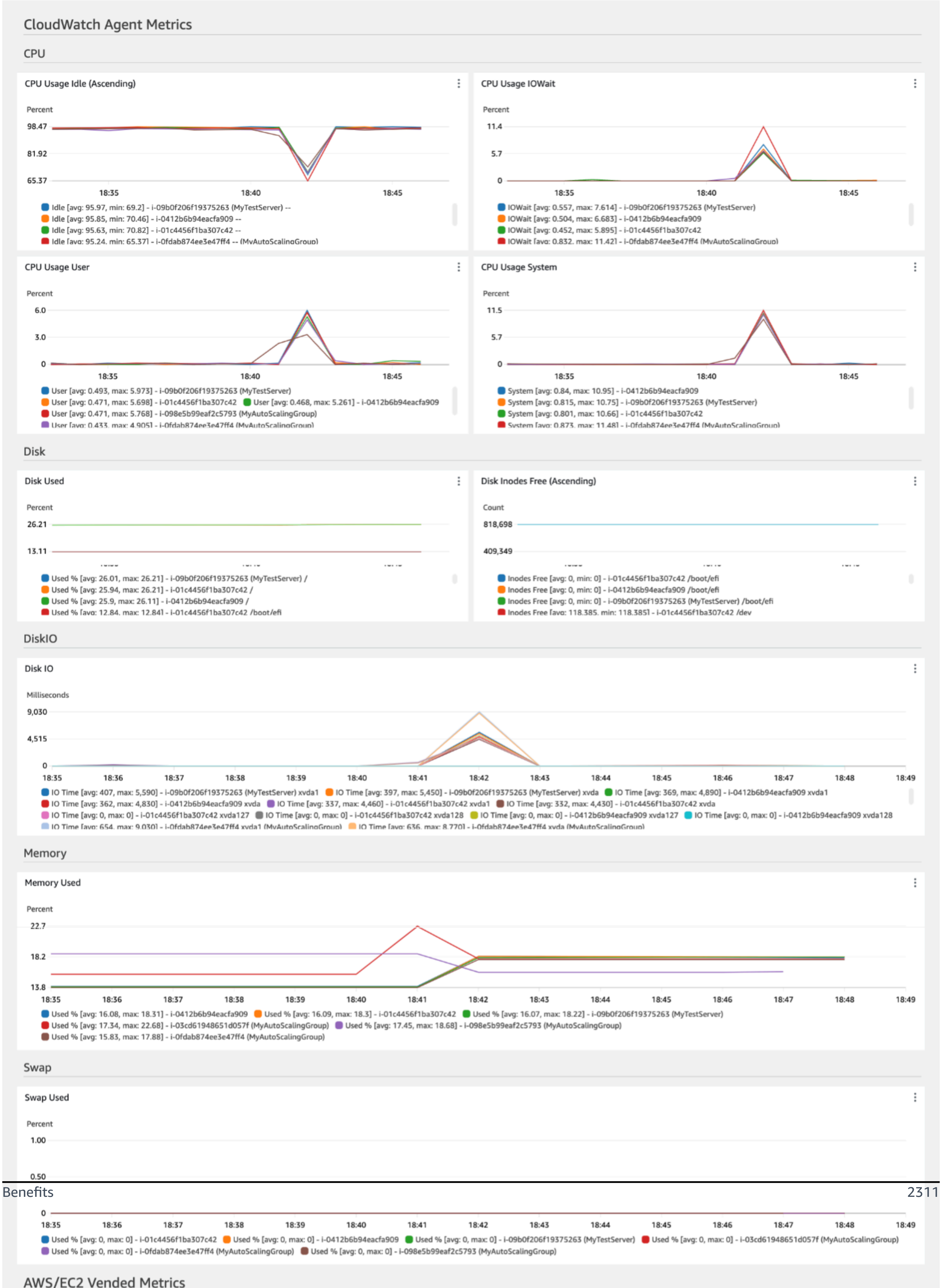
The solution delivers EC2 server monitoring using the CloudWatch Agent, providing additional system-level metrics on top of the standard EC2 namespace metrics for the following use cases:

- Detect CPU performance issues and resource constraints.
- Monitor disk utilization and storage capacity across different disks throughout your EC2 instances.
- Track memory usage patterns and potential memory leaks.
- Analyze I/O operations and their impact on overall performance.
- Observe network traffic patterns and potential anomalies.

Below are the key advantages of the solution:

- Automates metric collection for EC2 instances eliminating manual instrumentation.
- Provides a pre-configured, consolidated CloudWatch dashboard for EC2 instance metrics. The dashboard will automatically handle metrics from new EC2 instances configured using the solution, even if those metrics don't exist when you first create the dashboard. It also allows you to observe EC2 instances managed via Auto Scaling groups.

The following image is an example of the dashboard for this solution.


CloudWatch Agent Metrics
CPU
CPU Usage Idle (Ascending)
Percent
98.47
81.92
65.37
18:35
18:40
18:45
Idle [avg: 95.97, min: 69.2] - i-09b0f206f19375263 (MyTestServer) --
Idle [avg: 95.85, min: 70.46] - i-0412b6b94eacfa909 --
Idle [avg: 95.63, min: 70.82] - i-01c4456f1ba307c42 --
Idle [avg: 95.24, min: 65.37] - i-0fdab874ee3e47ff4 -- (MyAutoScalingGroup)
CPU Usage IOWait
Percent
11.4
5.7
0
18:35
18:40
18:45
IOWait [avg: 0.557, max: 7.614] - i-09b0f206f19375263 (MyTestServer)
IOWait [avg: 0.504, max: 6.683] - i-0412b6b94eacfa909
IOWait [avg: 0.452, max: 5.895] - i-01c4456f1ba307c42
IOWait [avg: 0.832, max: 11.42] - i-0fdab874ee3e47ff4 (MyAutoScalingGroup)
CPU Usage User
Percent
6.0
3.0
0
18:35
18:40
18:45
User [avg: 0.493, max: 5.973] - i-09b0f206f19375263 (MyTestServer)
User [avg: 0.471, max: 5.698] - i-01c4456f1ba307c42
User [avg: 0.468, max: 5.261] - i-0412b6b94eacfa909
User [avg: 0.471, max: 5.768] - i-098e5b99eaf2c5793 (MyAutoScalingGroup)
User [avg: 0.433, max: 4.905] - i-0fdab874ee3e47ff4 (MyAutoScalingGroup)
CPU Usage System
Percent
11.5
5.7
0
18:35
18:40
18:45
System [avg: 0.84, max: 10.95] - i-0412b6b94eacfa909
System [avg: 0.815, max: 10.75] - i-09b0f206f19375263 (MyTestServer)
System [avg: 0.801, max: 10.66] - i-01c4456f1ba307c42
System [avg: 0.873, max: 11.48] - i-0fdab874ee3e47ff4 (MyAutoScalingGroup)
Disk
Disk Used
Percent
26.21
13.11
Used % [avg: 26.01, max: 26.21] - i-09b0f206f19375263 (MyTestServer) /
Used % [avg: 25.94, max: 26.21] - i-01c4456f1ba307c42 /
Used % [avg: 25.9, max: 26.11] - i-0412b6b94eacfa909 /
Used % [avg: 12.84, max: 12.84] - i-01c4456f1ba307c42 /boot/efi
Disk Inodes Free (Ascending)
Count
818,698
409,349
Inodes Free [avg: 0, min: 0] - i-01c4456f1ba307c42 /boot/efi
Inodes Free [avg: 0, min: 0] - i-0412b6b94eacfa909 /boot/efi
Inodes Free [avg: 0, min: 0] - i-09b0f206f19375263 (MyTestServer) /boot/efi
Inodes Free [avg: 118.385, min: 118.385] - i-01c4456f1ba307c42 /dev
DiskIO
Disk IO
Milliseconds
9,030
4,515
0
18:35
18:36
18:37
18:38
18:39
18:40
18:41
18:42
18:43
18:44
18:45
18:46
18:47
18:48
18:49
IO Time [avg: 407, max: 5,590] - i-09b0f206f19375263 (MyTestServer) xvda1
IO Time [avg: 397, max: 5,450] - i-09b0f206f19375263 (MyTestServer) xvda
IO Time [avg: 369, max: 4,890] - i-0412b6b94eacfa909 xvda1
IO Time [avg: 362, max: 4,830] - i-0412b6b94eacfa909 xvda
IO Time [avg: 337, max: 4,460] - i-01c4456f1ba307c42 xvda1
IO Time [avg: 332, max: 4,430] - i-01c4456f1ba307c42 xvda
IO Time [avg: 0, max: 0] - i-01c4456f1ba307c42 xvda127
IO Time [avg: 0, max: 0] - i-01c4456f1ba307c42 xvda128
IO Time [avg: 0, max: 0] - i-0412b6b94eacfa909 xvda127
IO Time [avg: 0, max: 0] - i-0412b6b94eacfa909 xvda128
IO Time [avg: 654, max: 9,030] - i-0fdab874ee3e47ff4 xvda1 (MyAutoScalingGroup)
IO Time [avg: 636, max: 8,770] - i-0fdab874ee3e47ff4 xvda (MyAutoScalingGroup)
Memory
Memory Used
Percent
22.7
18.2
13.8
18:35
18:36
18:37
18:38
18:39
18:40
18:41
18:42
18:43
18:44
18:45
18:46
18:47
18:48
18:49
Used % [avg: 16.08, max: 18.31] - i-0412b6b94eacfa909
Used % [avg: 16.09, max: 18.3] - i-01c4456f1ba307c42
Used % [avg: 16.07, max: 18.22] - i-09b0f206f19375263 (MyTestServer)
Used % [avg: 17.34, max: 22.68] - i-03cd61948651d057f (MyAutoScalingGroup)
Used % [avg: 17.45, max: 18.68] - i-098e5b99eaf2c5793 (MyAutoScalingGroup)
Used % [avg: 15.83, max: 17.88] - i-0fdab874ee3e47ff4 (MyAutoScalingGroup)
Swap
Swap Used
Percent
1.00
0.50
0
18:35
18:36
18:37
18:38
18:39
18:40
18:41
18:42
18:43
18:44
18:45
18:46
18:47
18:48
18:49
Used % [avg: 0, max: 0] - i-01c4456f1ba307c42
Used % [avg: 0, max: 0] - i-0412b6b94eacfa909
Used % [avg: 0, max: 0] - i-09b0f206f19375263 (MyTestServer)
Used % [avg: 0, max: 0] - i-03cd61948651d057f (MyAutoScalingGroup)
Used % [avg: 0, max: 0] - i-0fdab874ee3e47ff4 (MyAutoScalingGroup)
Used % [avg: 0, max: 0] - i-098e5b99eaf2c5793 (MyAutoScalingGroup)
AWS/EC2 Vended Metrics

Costs

This solution creates and uses resources in your account. You are charged for standard usage, including the following:

- All metrics collected by the CloudWatch agent are charged as custom metrics. The number of metrics used by this solution depends on the number of EC2 hosts.

  The total number of CloudWatch agent metrics depends on the configuration of disks. Excluding disk and diskio metrics, the solution publishes six metrics. The number of disk metrics (`disk_used_percent`, `disk_inodes_free`) depend on the count of `device/fstype/path` dimensions. The diskio metric (`diskio_io_time`) depends on the count of `name` dimensions. For example, a single t2.micro with default settings as per EC2 console experience, produces a total of 22 CloudWatch agent metrics (4 CPU, 12 disk, 4 diskio, 1 memory, and 1 swap). Vended metrics like `AWS/EC2` are provided free of charge.
- One custom dashboard.
- API operations requested by the CloudWatch agent to publish the metrics. With the default configuration for this solution, the CloudWatch agent calls the **PutMetricData** once every minute. This means the **PutMetricData** API will be called `30*24*60=43,200` in a 30-day month for each EC2 host.

For more information about CloudWatch pricing, see [Amazon CloudWatch Pricing](#).

The pricing calculator can help you estimate approximate monthly costs for using this solution.

To use the pricing calculator to estimate your monthly solution costs

1. Open the [Amazon CloudWatch pricing calculator](#).
2. In the **Metrics** section, for **Number of metrics**, enter **`(6 + total count of disk and diskio metrics per EC2 host as described above) * number of EC2 instances configured for this solution`**.
3. In the **APIs** section, for **Number of API requests**, enter **`43200 * number of EC2 instances configured for this solution`**.
4. By default, the solution performs one **PutMetricData** operation each minute for each EC2 host.
5. In the **Dashboards and Alarms** section, for **Number of Dashboards**, enter **1**.
6. You can see your monthly estimated costs at the bottom of the pricing calculator.

CloudWatch agent configuration for this solution

The CloudWatch agent is software that runs continuously and autonomously on your servers and in containerized environments. It collects metrics, logs, and traces from your infrastructure and applications and sends them to CloudWatch and X-Ray.

For more information about the CloudWatch agent, see Collect metrics, logs, and traces with the CloudWatch agent.

The agent configuration in this solution collects a set of metrics to help you get started monitoring and observing your EC2 instances. The CloudWatch agent can be configured to collect more EC2 metrics than the dashboard displays by default. For a list of Amazon EC2 metrics, see Metrics collected by the CloudWatch agent on Linux and macOS instances. For information about metrics collected on Windows instances, see Metrics collected by the CloudWatch agent on Windows Server instances.

Agent configuration for EC2 Health solution

The metrics collected by the agent are defined in the agent configuration. The solution provides agent configurations to collect the recommended metrics with suitable dimensions for the solution's dashboard.

The steps for deploying the solution are described later in the section called "Deploy the agent for your solution". The following information is intended to help you understand how to customize the agent configuration for your environment.

> **Note**
>
> If an EC2 instance is not part of an Auto Scaling group, the CloudWatch agent drops the `AutoScalingGroupName` dimension entirely. This behavior helps to prevent dimension names with null/empty values. Each metric widget included in the solution dashboard searches for metrics which include and exclude the `AutoScalingGroup` dimension. This helps to ensure that all EC2 instances where the solution is applied are supported by the same dashboard.

If you wish to make any modifications to the agent configuration, you must apply the same changes to the solution's accompanying dashboard. For example, if you decide to omit the ImageId dimension, the same dimension must be removed from the metric search expression used in the dashboard widgets.

Agent configuration for EC2 Instances

Use the following CloudWatch agent configuration on Amazon EC2 instances where your workloads are deployed.

```
{
    "agent": {
      "metrics_collection_interval": 60,
      "run_as_user": "cwagent"
    },
    "metrics": {
      "append_dimensions": {
        "InstanceId": "${aws:InstanceId}",
        "InstanceType": "${aws:InstanceType}",
        "ImageId": "${aws:ImageId}",
        "AutoScalingGroupName": "${aws:AutoScalingGroupName}"
      },
      "metrics_collected": {
        "cpu": {
          "measurement": [
            "cpu_usage_idle",
             "cpu_usage_iowait",
             "cpu_usage_user",
             "cpu_usage_system"
          ],
          "totalcpu": true
        },
        "disk": {
          "measurement": [
            "used_percent",
            "inodes_free"
          ],
          "resources": [
            "*"
          ],
          "dimensions": [
            ["device", "fstype", "path"]
          ]
        },
        "diskio": {
```

```
            "measurement": [
              "io_time"
            ],
            "resources": [
              "*"
            ]
          },
        "mem": {
          "measurement": [
            "used_percent"
          ]
        },
        "swap": {
          "measurement": [
            "used_percent"
          ]
        }
      }
    }
  }
```

Deploy the agent for your solution

There are several approaches for installing the CloudWatch agent, depending on the use case. We recommend using Systems Manager for this solution. It provides a console experience and makes it simpler to manage a fleet of managed servers within a single Amazon account. The instructions in this section use Systems Manager and are intended for when you don't have the CloudWatch agent running with existing configurations. You can check whether the CloudWatch agent is running by following the steps in Verify that the CloudWatch agent is running.

If you are already running the CloudWatch agent on the EC2 hosts and managing the agent configurations, you can skip the instructions in this section and follow your existing deployment mechanism to update the configuration. Be sure to merge the EC2 Health agent configuration with your existing agent configuration, and then deploy the merged configuration. If you are using Systems Manager to store and manage the configuration for the CloudWatch agent, you can merge the configuration to the existing parameter value. For more information, see Managing CloudWatch agent configuration files.

> **Note**
>
> Using Systems Manager to deploy the following CloudWatch agent configurations will replace or overwrite any existing CloudWatch agent configuration on your EC2 instances. You can modify this configuration to suit your unique environment or use case. The metrics defined in configuration are the minimum required for the dashboard provided the solution.

The deployment process includes the following steps:

- Step 1: Ensure that the target EC2 instances have the required IAM permissions.
- Step 2: Store the recommended agent configuration file in the Systems Manager Parameter Store.
- Step 3: Install the CloudWatch agent on one or more EC2 instances using an Amazon CloudFormation stack.
- Step 4: Verify the agent setup is configured properly.

Step 1: Ensure the target EC2 instances have the required IAM permissions

You must grant permission for Systems Manager to install and configure the CloudWatch agent. You must also grant permission for the CloudWatch agent to publish telemetry from your EC2 instance to CloudWatch. Make sure that the IAM role attached to the instance has the **CloudWatchAgentServerPolicy** and **AmazonSSMManagedInstanceCore** IAM policies attached.

- To create a role, see Create IAM roles to use with the CloudWatch agent on Amazon EC2 instances.
- After the role is created, attach the role to your EC2 instances. To attach a role to an EC2 instance, follow the steps in Attach an IAM role to an instance.

Step 2: Store the recommended CloudWatch agent configuration file in Systems Manager Parameter Store

Parameter Store simplifies the installation of the CloudWatch agent on an EC2 instance by securely storing and managing configuration parameters, eliminating the need for hard-coded values. This ensures a more secure and flexible deployment process, enabling centralized management and easier updates to configurations across multiple instances.

Use the following steps to store the recommended CloudWatch agent configuration file as a parameter in Parameter Store.

To create the CloudWatch agent configuration file as a parameter

1. Open the Amazon Systems Manager console at https://console.aws.amazon.com/systems-manager/.
2. Verify that the selected Region on the console is the Region where the EC2 instances are running.
3. From the navigation pane, choose **Application Management**, **Parameter Store**.
4. Follow these steps to create a new parameter for the configuration.

    a. Choose **Create parameter**.

    b. In the **Name** box, enter a name that you'll use to reference the CloudWatch agent configuration file in later steps. For example, **`AmazonCloudWatch-EC2Health-Configuration`**.

    c. (Optional) In the **Description** box, type a description for the parameter.

    d. For **Parameter tier**, choose **Standard**.

    e. For **Type**, choose **String**.

    f. For **Data type**, choose **text**.

    g. In the **Value** box, paste the agent configuration JSON provided earlier in this document.

    h. Choose **Create parameter**.

Step 3: Install the CloudWatch agent and apply the configuration using an Amazon CloudFormation template

You can use Amazon CloudFormation to install the agent and configure it to use the CloudWatch agent configuration that you created in the previous steps.

To install and configure the CloudWatch agent for this solution

1. Open the Amazon CloudFormation **Quick create stack** wizard using this link: https:// console.aws.amazon.com/cloudformation/home?#/stacks/quickcreate?templateURL=https://

aws-observability-solutions-prod-us-east-1.s3.us-east-1.amazonaws.com/CloudWatchAgent/CFN/v1.0.0/cw-agent-installation-template-1.0.0.json .

2. Verify that the selected Region on the console is the Region where the EC2 instances are running.
3. For **Stack name**, enter a name to identity this stack, such as **`CWAgentInstallationStack`**.
4. In the **Parameters** section, specify the following:

   a. For **CloudWatchAgentConfigSSM**, enter the name of the Systems Manager parameter for the agent configuration that you created earlier, such as **`AmazonCloudWatch-EC2Health-Configuration`**.

   b. To select the target instances, you have two options.

      i. For **InstanceIds**, specify a comma-delimited list of instance IDs list of instance IDs where you want to install the CloudWatch agent with this configuration. You can list a single instance or several instances.

      ii. If you are deploying at scale, you can specify the **TagKey** and the corresponding **TagValue** to target all EC2 instances with this tag and value. If you specify a **TagKey**, you must specify a corresponding **TagValue**. (For an Auto Scaling group, specify **`aws:autoscaling:groupName`** for the **TagKey** and specify the Auto Scaling group name for the **TagValue** to deploy to all instances within the Auto Scaling group.)

   If you specify both the **InstanceIds** and the **TagKeys** parameters, the **InstanceIds** will take precedence and the tags will be ignored.

5. Review the settings, then choose **Create stack**.

If you want to edit the template file first to customize it, choose the **Upload a template file** option under **Create Stack Wizard** to upload the edited template. For more information, see Creating a stack on Amazon CloudFormation console. You can use the following link to download the template: https://aws-observability-solutions-prod-us-east-1.s3.us-east-1.amazonaws.com/CloudWatchAgent/CFN/v1.0.0/cw-agent-installation-template-1.0.0.json .

> **Note**
>
> After this step is completed, this Systems Manager parameter will be associated with the CloudWatch agents running in the targeted instances. This means that:

1. If the Systems Manager parameter is deleted, the agent will stop.
2. If the Systems Manager parameter is edited, the configuration changes will automatically apply to the agent at the scheduled frequency which is 30 days by default.
3. If you want to immediately apply changes to this Systems Manager parameter, you must run this step again. For more information about associations, see Working with associations in Amazon Systems Manager.

Step 4: Verify the agent setup is configured properly

You can verify whether the CloudWatch agent is installed by following the steps in Verify that the CloudWatch agent is running. If the CloudWatch agent is not installed and running, make sure you have set up everything correctly.

- Be sure you have attached a role with correct permissions for the EC2 instance as described in the section called "Step 1: Ensure the target EC2 instances have the required IAM permissions".
- Be sure you have correctly configured the JSON for the Systems Manager parameter. Follow the steps in Troubleshooting installation of the CloudWatch agent with Amazon CloudFormation.

To verify that EC2 health metrics are being published to CloudWatch

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. Choose **Metrics**, **All metrics**.
3. Make sure you've selected the Region where you deployed the solution, and choose **Custom namespaces**, **CWAgent**.
4. Search for the metrics mentioned in the agent configuration section of this document, such as `mem_used_percent`. If you see results for these metrics, then the metrics are being published to CloudWatch.

Create the EC2 Health solution dashboard

This dashboard displays the newly emitted metrics, showing the EC2 Health metrics. This dashboard provides a top contributor view for the health of your EC2 instances in a single region. The top contributor view displays the top 10 per metric widget. This allows you to identify outliers at a glance.

To create the dashboard, you can use the following options:

- Use CloudWatch console to create the dashboard.
- Use Amazon CloudFormation console to deploy the dashboard.
- Download the Amazon CloudFormation infrastructure as code and integrate it as part of your continuous integration (CI) automation.

By using the CloudWatch console to create a dashboard, you can preview the dashboard before actually creating and being charged.

> **Note**
>
> The dashboard created with Amazon CloudFormation in this solution displays metrics from the Region where the solution is deployed. Be sure to create the Amazon CloudFormation stack in the Region where your EC2 metrics are published.
> If you've specified a custom namespace other than `CWAgent` in the CloudWatch agent configuration, you'll have to change the Amazon CloudFormation template for the dashboard to replace `CWAgent` with the customized namespace you are using.

To create the dashboard via CloudWatch Console

1. Open the CloudWatch Console **Create Dashboard** using this link: https://console.aws.amazon.com/cloudwatch/home?#dashboards?dashboardTemplate=Ec2LinuxMacOsHealth&referrer=os-catalog .
2. Verify that the selected Region on the console is the Region where the EC2 instances are running.
3. Enter the name of the dashboard, then choose **Create Dashboard**.

   To easily differentiate this dashboard from similar dashboards in other Regions, we recommend including the Region name in the dashboard name, such as **`EC2HealthDashboard-us-east-1`**.
4. Preview the dashboard and choose **Save** to create the dashboard.

To create the dashboard via Amazon CloudFormation

1. Open the Amazon CloudFormation **Quick create stack** wizard using this link: https://console.aws.amazon.com/cloudformation/home?#/stacks/quickcreate?templateURL=https://aws-observability-solutions-prod-us-east-1.s3.us-east-1.amazonaws.com/EC2_Health/CloudWatch/CFN/v1.0.0/dashboard-template-linux-macos-1.0.0.json .
2. Verify that the selected Region on the console is the Region where the EC2 instances are running.
3. For **Stack name**, enter a name to identity this stack, such as `EC2HealthDashboardStack`.
4. In the **Parameters** section, specify the name of the dashboard under the **DashboardName** parameter.

   To easily differentiate this dashboard from similar dashboards in other Regions, we recommend including the Region name in the dashboard name, such as **`EC2HealthDashboard-us-east-1`**.
5. Acknowledge access capabilities for transforms under **Capabilities and transforms**. Note that Amazon CloudFormation doesn't add any IAM resources.
6. Review the settings, then choose **Create stack**.
7. After the stack status is **CREATE_COMPLETE**, choose the **Resources** tab under the created stack and then choose the link under **Physical ID** to go to the dashboard. You can also access the dashboard in the CloudWatch console by choosing **Dashboards** in the left navigation pane of the console, and finding the dashboard name under **Custom Dashboards**.

If you want to edit the template file to customize it for any purpose, you can use the **Upload a template file** option under **Create Stack Wizard** to upload the edited template. For more information, see Creating a stack on Amazon CloudFormation console. You can use this link to download the template: https://aws-observability-solutions-prod-us-east-1.s3.us-east-1.amazonaws.com/EC2_Health/CloudWatch/CFN/v1.0.0/dashboard-template-linux-macos-1.0.0.json

Get started with the EC2 Health solution dashboard

Here are a few tasks that you can try out with the new EC2 monitoring dashboard. These tasks allow you to validate that the dashboard is working correctly and provide you some hands-on experience using it to monitor EC2 instances. As you try these out, you'll get familiar with navigating the dashboard and interpreting the visualized metrics.

Monitor the various CPU utilization metrics

In the **CPU** section, examine the array of CPU usage metrics. These provide insight into how CPU resources are being utilized across different activities like user processes, system tasks, and I/O operations. Look for instances with consistently high utilization or unusual patterns, which might indicate the need for scaling or optimization.

Analyze disk utilization across different devices

Navigate to the **Disk** section to find the storage usage and inode availability metrics. These help you identify instances that are running low on storage space or file system resources. Pay attention to any instances approaching high disk usage levels, as this could lead to performance issues or service disruptions.

Investigate memory utilization patterns

In the **Memory** section, observe the graph which plots memory utilization over time. This shows how much of the available memory is being used by each instance. Look for patterns or spikes in memory usage that might correlate with specific times or events. High memory utilization could indicate the need for instance resizing or application optimization.

Correlate patterns across core utilization metrics

Compare and watch out for related utilization patterns. For example, a workload running a log rotation process could present regular increases in **CPU** and **memory** utilization, followed by a decrease in **disk** utilization.

Inspect network activity

In the **Network** section, examine the inbound and outbound network traffic metrics, both in terms of data volume and packet count. These give you insight into the network activity for your EC2 instances. Look out for both regular or anomalous spikes in network traffic, or imbalances between inbound and outbound data.

Monitor across accounts and Regions

To enable unified monitoring across accounts, CloudWatch offers the following features:

- **CloudWatch cross-account observability**– facilitate observability within a single Region with the Observability Access Manager (OAM) service. You can link accounts and easily view metrics, logs, traces, and other telemetry between accounts. This helps you to unify observability in central monitoring accounts that view telemetry shared from source accounts, and operate on this shared telemetry as if it were native to the monitoring account.
- **Cross-account cross-Region CloudWatch console**– delivers a console experience that allows you to view dashboards, metrics, and alarms consoles of other accounts across Regions by toggling between accounts. After you set up the necessary permissions, you use an account selector integrated into the alarms, dashboards, and metrics consoles to view metrics, dashboards, and alarms in other accounts without having to log in and out of the accounts. By enabling this feature, you can also set up dashboards that contain cross-account cross-Region metrics for centralized visibility within an account.

These two features are complementary to each other and can be used independently or together. See the following table for a comparison of the two features. We recommend that you use CloudWatch cross-account observability for the richest cross-account observability and discovery experience within a Region for your metrics, logs, and traces.

	CloudWatch cross-account observability	Cross-account cross-Region CloudWatch console
What is it?	Unified access to underlying telemetry and other observabi lity resources across multiple accounts. After this is configure d, observability resources are seamlessly viewable between accounts, eliminating the need for role assumptions. The central monitoring account gains direct access to the telemetry data and resources from source accounts,	A designated monitoring account assume a **CrossAccountSharin gRole** defined in source accounts from the CloudWatch console. By assuming this role, the monitorin g account can invoke operation s such as dashboard viewing on behalf of source accounts, directly from its console.

	[CloudWatch cross-account observability]()	[Cross-account cross-Region CloudWatch console]()
	streamlining the monitoring and observability process.	
How does it work?	A monitoring account, using the Observability Access Monitorin g service, creates a *sink* and attaches a sink policy to it. The sink policy defines which resources they would like to view and which source accounts should share them. Then source accounts can create a link to the monitoring account sink, establish ing what they actually want to share. After the link is created, the specified resources are visible in the monitoring account.	A source account initiates the configuration by setting up a **CrossAccountSharingRole**, allowing a monitoring account to run operations in the source account. Then, a monitoring account enables the cross-acc ount cross-Region selector in the console by specifying the source account ID. This enables the monitoring account to be able to switch into the source account. When switching, the CloudWatch console checks for the existence of a service-linked role that allows CloudWatch to assume the **CrossAccountSharin gRole** that was created in the source account.
What telemetry is supported?	• Metrics • Traces • Logs .	• Metrics • Traces

	CloudWatch cross-account observability	Cross-account cross-Region CloudWatch console
What functionality is supported?	• Dashboarding • Alarms • Metrics Insights • Anomaly Detection • CloudWatch Logs insights • Application Insights • Other functionalities. For more details see CloudWatch cross-account observability. .	• Console switching between accounts and Regions in metrics, alarms, and traces consoles. • Custom dashboards with metrics and alarms from other accounts and Regions For more details, see Cross-account cross-Region CloudWatch console.
How many accounts can I use it with?	A monitoring account can see resources from as many as 100,000 source accounts at the same time. A source account can share their resources with as many as five different monitoring accounts.	By using the cross-account cross-Region selector in the console, a monitoring account can switch to one other account at a time but there is no limit on the number of accounts that can be linked. When defining cross-account dashboards and alarms, many source accounts can be reference d.
Does it move telemetry data?	No. Resources are shared between accounts with the exception of copied traces.	No. An IAM policy is configure d to allow embedded account switching for cross-account cross-Region resource visibilty.

	CloudWatch cross-account observability	Cross-account cross-Region CloudWatch console
How much does it cost?	No extra charges for shared logs and metrics, and the first trace copy is free. For more informati on about pricing, see Amazon CloudWatch pricing.	No additional charges for cross-account or cross-Region actions.
Does it support observability across Regions?	No	Yes
Does it support programmatic access?	Yes. the Amazon CLI, Amazon Cloud Development Kit (Amazon CDK), and APIs are supported.	No.
Does it support programmatic setup?	Yes	Yes
Does it support Amazon Organizat ions?	Yes	Yes

Topics

- CloudWatch cross-account observability
- Cross-account cross-Region CloudWatch console

CloudWatch cross-account observability

With Amazon CloudWatch cross-account observability, you can monitor and troubleshoot applications that span multiple accounts within a Region. Seamlessly search, visualize, and analyze your metrics, logs, traces, Application Signals services and service level objectives (SLOs), Application Insights applications, and internet monitors in any of the linked accounts without account boundaries.

Set up one or more Amazon accounts as *monitoring accounts* and link them with multiple *source accounts*. A monitoring account is a central Amazon account that can view and interact with observability data generated from source accounts. A source account is an individual Amazon account that generates observability data for the resources that reside in it. Source accounts share their observability data with the monitoring account. The shared observability data can include the following types of telemetry:

- Metrics in Amazon CloudWatch. You can choose to share the metrics from all namespaces with the monitoring account, or filter to a subset of namespaces.
- Log groups in Amazon CloudWatch Logs. You can choose to share all log groups with the monitoring account, or filter to a subset of log groups.
- Traces in Amazon X-Ray
- Services and Service level objectives (SLOs) in Application Signals
- Applications in Amazon CloudWatch Application Insights
- Monitors in CloudWatch Internet Monitor

To create links between monitoring accounts and source accounts, you can use the CloudWatch console. Alternatively, use the *Observability Access Manager* commands in the Amazon CLI and API. For more information, see Observability Access Manager API Reference.

A *sink* is a resource that represents an attachment point in a monitoring account. Source accounts can link to the sink to share observability data. Each account can have one sink per Region. Each sink is managed by the monitoring account where it is located. An *observability link* is a resource that represents the link established between a source account and a monitoring account. Links are managed by the source account.

The next topic explains how to set up CloudWatch cross-account observability in both monitoring accounts and source accounts. For information about the cross-account cross-Region CloudWatch dashboard, see Cross-account cross-Region CloudWatch console.

Use Organizations for source accounts

There are two options for linking source accounts to your monitoring account. You can use one or both options.

- Use Amazon Organizations to link accounts in an organization or organizational unit to the monitoring account.

- Connect individual Amazon accounts to the monitoring account.

We recommend that you use Organizations so that new Amazon accounts created later in the organization are automatically onboarded to cross-account observability as source accounts.

Details about linking monitoring accounts and source accounts

- Each monitoring account can be linked to as many as 100,000 source accounts.
- Each source account can share data with as many as five monitoring accounts.
- You can set up a single account as both a monitoring account and a source account. If you do, this account sends only the observability data from itself to its linked monitoring account. It does not relay the data from its source accounts.
- A monitoring account specifies which telemetry types can be shared with it. A source account specifies which telemetry types it wants to share.
  - If there are more telemetry types selected in the *monitoring account* than in the source account, the accounts are linked. Only the data types that are selected in both accounts are shared.
  - If there are more telemetry types selected in the *source account* than in the monitoring account, the link creation fails and nothing is shared.
  - A metric name doesn't appear in the monitoring account console until that metric emits new data points after the link is created.
- To remove a link between accounts, do so from the source account.
- To delete a sink in a monitoring account, you must first remove all links to that sink the monitoring account.

Pricing

Cross-account observability in CloudWatch comes with no extra cost for logs and metrics, Application Signals, and the first trace copy is free. For more information about pricing, see [Amazon CloudWatch Pricing](#).

Contents

Link monitoring accounts with source accounts

The topics in this section explain how to set up links between monitoring accounts and source accounts.

We recommend that you create a new Amazon account to serve as the monitoring account for your organization.

Contents

- Use a URL to set up individual source accounts

Necessary permissions

Permissions needed to create links

To create links between a monitoring account and a source account, you must be signed in with certain permissions.

- **To set up a monitoring account –** You must have either full administrator access in the monitoring account, or you must sign in to that account with the following permissions:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowSinkModification",
            "Effect": "Allow",
            "Action": [
                "oam:CreateSink",
                "oam:DeleteSink",
                "oam:PutSinkPolicy",
                "oam:TagResource"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AllowReadOnly",
            "Effect": "Allow",
            "Action": ["oam:Get*", "oam:List*"],
            "Resource": "*"
        }
    ]
}
```

- **Source account, scoped to a specific monitoring account –** To create, update, and manage links for just one specified monitoring account, you must sign in to account with at least the following permissions. In this example, the monitoring account is 999999999999.

  If the link isn't going to share all seven resource types (metrics, logs, traces, Application Insights applications, Application Signals services and service level objectives (SLOs), and

Internet Monitor monitors), you can omit `cloudwatch:Link`, `logs:Link`, `xray:Link`, `applicationinsights:Link`, `application-signals:Link`, or `internetmonitor:Link` as needed.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "oam:CreateLink",
                "oam:UpdateLink",
                "oam:DeleteLink",
                "oam:GetLink",
                "oam:TagResource"
            ],
            "Effect": "Allow",
            "Resource": "arn:*:oam:*:*:link/*"
        },
        {
            "Action": [
                "oam:CreateLink",
                "oam:UpdateLink"
            ],
            "Effect": "Allow",
            "Resource": "arn:*:oam:*:*:sink/*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceAccount": [
                        "999999999999"
                    ]
                }
            }
        },
        {
            "Action": "oam:ListLinks",
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Action": "cloudwatch:Link",
            "Effect": "Allow",
            "Resource": "*"
        },
```

```
        {
            "Action": "logs:Link",
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Action": "xray:Link",
            "Effect": "Allow",
            "Resource": "*"
        },
        {
             "Action": "applicationinsights:Link",
             "Effect": "Allow",
             "Resource": "*"
         },
        {
             "Action": "internetmonitor:Link",
             "Effect": "Allow",
             "Resource": "*"
        },
        {
             "Action": "application-signals:Link",
             "Effect": "Allow",
             "Resource": "*"
        }
    ]
}
```

- **Source account, with permissions to link to any monitoring account** – To create a link to any existing monitoring account sink and share metrics, log groups, traces, Application Insights applications, and Internet Monitor monitors, you must sign in to the source account with full administrator permissions or sign in there with the following permissions

  If the link isn't going to share all seven resource types (metrics, logs, traces, Application Insights applications, Application Signals services and service level objectives (SLOs), and Internet Monitor monitors), you can omit `cloudwatch:Link`, `logs:Link`, `xray:Link`, `applicationinsights:Link`, `application-signals:Link`, or `internetmonitor:Link` as needed.

```
{
    "Version": "2012-10-17",
    "Statement": [{
```

```
            "Effect": "Allow",
            "Action": [
                "oam:CreateLink",
                "oam:UpdateLink"
            ],
            "Resource": [
                "arn:aws:oam:*:*:link/*",
                "arn:aws:oam:*:*:sink/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "oam:List*",
                "oam:Get*"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "oam:DeleteLink",
                "oam:GetLink",
                "oam:TagResource"
            ],
            "Resource": "arn:aws:oam:*:*:link/*"
        },
        {
            "Action": "cloudwatch:Link",
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Action": "xray:Link",
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Action": "logs:Link",
            "Effect": "Allow",
            "Resource": "*"
        },
        {
             "Action": "applicationinsights:Link",
```

```
                "Effect": "Allow",
                "Resource": "*"
            },
            {
                "Action": "internetmonitor:Link",
                "Effect": "Allow",
                "Resource": "*"
            },
            {
                "Action": "application-signals:Link",
                "Effect": "Allow",
                "Resource": "*"
            }
        ]
}
```

Permissions needed to monitor across accounts

After a link has been created, to view source account information from a monitoring account, you must be signed in to an account with one of the following:

- Full administrator access in the monitoring account
- The following cross-account permissions, in addition to permissions to view the specific types of resources that you will be monitoring

```
{
   "Sid": "AllowReadOnly",
   "Effect": "Allow",
   "Action": [
     "oam:Get*",
     "oam:List*"
   ],
   "Resource": "*"
 }
```

Setup overview

The following high-level steps show you how to set up CloudWatch cross-account observability.

> **Note**
>
> We recommend creating a new Amazon account to use as your organization's monitoring account.

1. Set up a dedicated monitoring account.
2. (Optional) Download an Amazon CloudFormation template or copy a URL to link source accounts.
3. Link source accounts to the monitoring account.

After completing these steps, you can use the monitoring account to view the observability data of the source accounts.

Step 1: Set up a monitoring account

Follow the steps in this section to set up an Amazon account as a monitoring account for CloudWatch cross-account observability.

Prerequisites

- **If you're setting up accounts in an Amazon Organizations organization as the source accounts** – Get the organization path or organization ID.
- **If you're not using Organizations for the source accounts** – Get the account IDs of the source accounts.

To set up an account as a monitoring account, you must have certain permissions. For more information, see Necessary permissions.

To set up a monitoring account

1. Sign in to the account that you want to use as a monitoring account.
2. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
3. In the left navigation pane, choose **Settings**.
4. By **Monitoring account configuration**, choose **Configure**.
5. For **Select data**, choose whether this monitoring account will be able to view **Logs**, **Metrics**, **Traces**, **Application Insights - Applications**, **Internet Monitor - Monitors**, and **Application**

Signals - Services, Service Level Objectives (SLOs) data from the source accounts it is linked to.

6. For **List source accounts**, enter the source accounts that this monitoring account will view. To identify the source accounts, enter individual account IDs, organization paths, or organization IDs. If you enter an organization path or organization ID, this monitoring account is allowed to view observability data from all linked accounts in that organization.

   Separate the entries in this list with commas.

   > **Important**
   >
   > When you enter an organization path, follow the exact format. The ou-id must end with a / (a slash character). For example: `o-a1b2c3d4e5/r-f6g7h8i9j0example/ou-def0-awsbbbb/`

7. For **Define a label to use to identify your source account**, you can define alabel that is used to create a Amazon CloudFormation template. The label is then applied to source accounts when that template is used to link the source accounts to this monitoring account.

   You can specify whether to use account names or email addresses in this label, and also use variables such as `$AccountName`, `$AcccountEmail`, and `$AcccountEmailNoDomain`.

   > **Note**
   >
   > In the Amazon GovCloud (US-East) and Amazon GovCloud (US-West) Regions, the only supported option is to use custom labels, and the `$AccountName`, `$AcccountEmail`, and `$AcccountEmailNoDomain` variables all resolve as *`account-id`* instead of the specified variable.

8. Choose **Configure**.

> **Important**
>
> The link between the monitoring and source accounts is not complete until you configure the source accounts. For more information, see the following sections.

Step 2: (Optional) Download an Amazon CloudFormation template or URL

To link source accounts to a monitoring account, we recommend using an Amazon CloudFormation template or a URL.

- **If you are linking an entire organization –** CloudWatch provides an Amazon CloudFormation template.
- **If you are linking individual accounts –** Use either an Amazon CloudFormation template or a URL that CloudWatch provides.

To use an Amazon CloudFormation template, you must download it during these steps. After you link the monitoring account with at least one source account, the Amazon CloudFormation template is no longer available to download.

To download an Amazon CloudFormation template or copy a URL for linking source accounts to the monitoring account

1. Sign in to the account that you want to use as a monitoring account.
2. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
3. In the left navigation pane, choose **Settings**.
4. By **Monitoring account configuration**, choose **Resources to link accounts**.
5. Do one of the following:
   - Choose **Amazon organization** to get a template to use to link accounts in an organization to this monitoring account.
   - Choose **Any account** to get a template or URL for setting up individual accounts as source accounts.
6. Do one of the following:
   - If you chose **Amazon organization**, choose **Download CloudFormation template**.
   - If you chose **Any account**, choose either **Download CloudFormation template** or **Copy URL**.
7. (Optional) Repeat steps 5-6 to download both the Amazon CloudFormation template and the URL.

Step 3: Link the source accounts

Use the steps in these sections to link source accounts to a monitoring account.

To link monitoring accounts with source accounts, you must have certain permissions. For more information, see Necessary permissions.

Use an Amazon CloudFormation template to set up all accounts in an organization or an organizational unit as source accounts

These steps assume that you already downloaded the necessary Amazon CloudFormation template by performing the steps in Step 2: (Optional) Download an Amazon CloudFormation template or URL.

To use an Amazon CloudFormation template to link accounts in an organization or organizational unit to the monitoring account

1. Sign in to the organization's management account.
2. Open the Amazon CloudFormation console at https://console.amazonaws.cn/cloudformation.
3. In the left navigation bar, choose **StackSets**.
4. Check that you are signed in to the Region that you want, then choose **Create StackSet**.
5. Choose **Next**.
6. Choose **Template is ready** and choose **Upload a template file**.
7. Choose **Choose file**, choose the template that you downloaded from the monitoring account, and choose **Open**.
8. Choose **Next**.
9. For **Specify StackSet details**, enter a name for the StackSet and choose **Next**.
10. For **Add stacks to stack set**, choose **Deploy new stacks**.
11. For **Deployment targets**, choose whether to deploy to the entire organization or to specified organizational units.
12. For **Specify regions**, choose which Regions to deploy CloudWatch cross-account observability to.
13. Choose **Next**.
14. On the **Review** page, confirm your selected options and choose **Submit**.
15. In the **Stack instances** tab, refresh the screen until you see that your stack instances have the status **CREATE_COMPLETE**.

Use an Amazon CloudFormation template to set up individual source accounts

These steps assume that you already downloaded the necessary Amazon CloudFormation template by performing the steps in Step 2: (Optional) Download an Amazon CloudFormation template or URL.

To use an Amazon CloudFormation template to set up individual source accounts for CloudWatch cross-account observability

1. Sign in to the source account.
2. Open the Amazon CloudFormation console at https://console.amazonaws.cn/cloudformation.
3. In the left navigation bar, choose **Stacks**.
4. Check that you are signed in to the Region that you want, then choose **Create stack**, **With new resources (standard)**.
5. Choose **Next**.
6. Choose **Upload a template file**.
7. Choose **Choose file**, choose the template that you downloaded from the monitoring account, and choose **Open**.
8. Choose **Next**.
9. For **Specify stack details**, enter a name for the stack and choose **Next**.
10. On the **Configure stack options** page, choose **Next**.
11. On the **Review** page, choose **Submit**.
12. On the status page for your stack, refresh the screen until you see that your stack has the status **CREATE_COMPLETE**.
13. To use this same template to link more source accounts to this monitoring account, sign out of this account and sign in to the next source account. Then repeat steps 2-12.

Use a URL to set up individual source accounts

These steps assume that you already copied the necessary URL by performing the steps in Step 2: (Optional) Download an Amazon CloudFormation template or URL.

To use a URL to link individual source accounts to the monitoring account

1. Sign in to the account that you want to use as a source account.
2. Enter the URL that you copied from the monitoring account.

You see the CloudWatch settings page, with some information filled in.

3. For **Select data**, choose whether this source account will share **Logs**, **Metrics**, **Traces**, **Application Insights - Applications**, and **Internet Monitor - Monitors** data to this monitoring account.

   For both **Logs** and **Metrics**, you can choose whether to share all resources or a subset with the monitoring account.

   a. (Optional) To share a subset of this account's log groups with the monitoring account, select **Logs** and choose **Filter Logs**. Then use the **Filter Logs** box to construct a query to find the log groups that you want to share. The query will use the term `LogGroupName` and one or more of the following operands.

      - `=` and `!=`
      - `AND`
      - `OR`
      - `^` indicates LIKE and `!^` indicates NOT LIKE. These can be used only as prefix searches. Include a `%` at the end of the string that you want to search for and include.
      - `IN` and `NOT IN`, using parentheses `( )`

      The complete query must be no more than 2000 characters and is limited to five conditional operands. Conditional operands are `AND` and `OR`. There isn't a limit on the number of other operands.

      > **Tip**
      >
      > Choose **View sample queries** to see the correct syntax for common query formats.

   b. (Optional) To share a subset of this account's metric namespaces with the monitoring account, select **Metrics** and choose **Filter Metrics**. Then use the **Filter Metrics** box to construct a query to find the metric namespaces that you want to share. Use the term `Namespace` and one or more of the following operands.

      - `=` and `!=`
      - `AND`
      - `OR`

- `LIKE` and `NOT LIKE`. These can be used only as prefix searches. Include a `%` at the end of the string that you want to search for and include.
- `IN` and `NOT IN`, using parentheses `( )`

The complete query must be no more than 2000 characters and is limited to five conditional operands. Conditional operands are `AND` and `OR`. There isn't a limit on the number of other operands.

> **Tip**
>
> Choose **View sample queries** to see the correct syntax for common query formats.

4. Do not change the ARN in **Enter monitoring account configuration ARN**.
5. The **Define a label to identify your source account** section is pre-filled with the label choice from the monitoring account, if there is one. Optionally, choose **Edit** to change it.

> **Note**
>
> In the Amazon GovCloud (US-East) and Amazon GovCloud (US-West) Regions, the only supported option is to use custom labels, and the `$AccountName`, `$AcccountEmail`, and `$AcccountEmailNoDomain` variables all resolve as *`account-id`* instead of the specified variable.

6. Choose **Link**.
7. Enter **`Confirm`** in the box and choose **Confirm**.
8. To use this same URL to link more source accounts to this monitoring account, sign out of this account and sign in to the next source account. Then repeat steps 2-7.

Manage monitoring accounts and source accounts

After you set up your monitoring accounts and source accounts, you can use the steps in these sections to manage them.

Contents

- Remove the link between a monitoring account and source account
- View information about a monitoring account

Link more source accounts to an existing monitoring account

Follow the steps in this section to add links from additional source accounts to an existing monitoring account.

Each source account can be linked to as many as five monitoring accounts. Each monitoring account can be linked to as many as 100,000 source accounts.

To manage a source account, you must have certain permissions. For more information, see Necessary permissions.

To add more source accounts to a monitoring account

1. Sign in to the monitoring account.
2. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
3. In the left navigation pane, choose **Settings**.
4. By **Monitoring account configuration**, choose **Manage source accounts**.
5. Choose the **Configuration policy** tab.
6. In the **Configuration policy** box, add the new source account ID in the **Principal** line.

   For example, suppose the **Principal** line is currently the following:

   ```
   "Principal": {"AWS": ["111111111111", "222222222222"]}
   ```

   To add 999999999999 as a third source account, edit the line to the following:

   ```
   "Principal": {"AWS": ["111111111111", "222222222222", "999999999999"]}
   ```
7. Choose **Update**.
8. Choose the **Configuration details** tab.
9. Choose the copy icon that is next to the monitoring account's sink ARN.
10. Sign in to the account that you want to use as a new source account.
11. Paste the monitoring account's sink ARN that you copied in Step 9.

You see the CloudWatch settings page, with some information filled in.

12. For **Select data**, choose whether this source account will send **Logs**, **Metrics**, **Traces**, and **Application Insights - Applications**, **Internet Monitor - Monitors**, and **Application Signals - Services, Service Level Objectives (SLOs)** data to the monitoring accounts it is linked to.
13. Do not change the ARN in **Enter monitoring account configuration ARN**.
14. The **Define a label to identify your source account** section is pre-filled with the label choice from the monitoring account, if there is one. Optionally, choose **Edit** to change it.

> **Note**
>
> In the Amazon GovCloud (US-East) and Amazon GovCloud (US-West) Regions, the only supported option is to use custom labels, and the `$AccountName`, `$AcccountEmail`, and `$AcccountEmailNoDomain` variables all resolve as *`account-id`* instead of the specified variable.

15. Choose **Link**.
16. Enter **`Confirm`** in the box and choose **Confirm**.

Remove the link between a monitoring account and source account

Follow the steps in this section to stop sending data from one source account to a monitoring account.

You must have the permissions required to manage a source account to complete this task. For more information, see Necessary permissions.

To remove the link between a source account and a monitoring account

1. Sign in to the source account.
2. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
3. In the left navigation pane, choose **Settings**.
4. By **Source account information**, choose **View monitoring accounts**.
5. Select the check box next to the monitoring account that you want to stop sharing data with.
6. Choose **Stop sharing data**, **Confirm**.
7. Sign in to the monitoring account.

8. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
9. Choose **Settings**.
10. By **Monitoring account information**, choose **View configuration**.
11. In the **Policy** box, delete the source account ID from the **Principal** line and choose **Update**.

View information about a monitoring account

Follow the steps in this section to view the cross-account settings for a monitoring account.

To manage a monitoring account, you must have certain permissions. For more information, see Necessary permissions.

To manage a monitoring account

1. Sign in to the monitoring account.
2. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
3. In the left navigation pane, choose **Settings**.
4. By **Monitoring account configuration**, choose **Manage source accounts**.
5. To view the Observability Access Manager policy that enables this account to be a monitoring account, choose the **Configuration policy** tab.
6. To view the source accounts that are linked to this monitoring account, choose the **Linked source accounts** tab.
7. To view the monitoring account sink ARN, and the types of data that this monitoring account can view in linked source accounts, choose the **Linked source accounts** tab.

Cross-account cross-Region CloudWatch console

> **Note**
>
> We recommend that you use CloudWatch cross-account observability to get the richest cross-account observability and discovery experience for your metrics, logs, and traces within a Region. For more information, see CloudWatch cross-account observability.

The cross-account, cross-Region CloudWatch console allows you to easily switch between different accounts and Region by using selectors in the console to view the dashboards, alarms, and metrics in other accounts and Regions. This feature also allows you to create cross-account, cross-Region dashboards which summarize your CloudWatch metrics from multiple Amazon accounts and multiple Regions into a single dashboard, making them accessible without having to switch accounts or Regions.

Many organizations have their Amazon resources deployed in multiple accounts, to provide billing and security boundaries. In this case, we recommend that you designate one or more of your accounts as your *monitoring accounts*, and build your cross-account cross-Region dashboards in these accounts. Cross-account cross-Region console functionality is integrated with Amazon Organizations, to help you efficiently build your cross-account cross-Region dashboards.

The cross-account, cross-region CloudWatch console experience does not provide cross-account cross-Region visibility for logs. Additionally, it does not support the creation of alarms on metrics in other accounts or Regions from within a monitoring account.

Topics

- Enabling cross-account cross-Region functionality in CloudWatch
- (Optional) Integrate with Amazon Organizations
- Troubleshooting your CloudWatch cross-account setup
- Disabling and cleaning up after using cross-account

Enabling cross-account cross-Region functionality in CloudWatch

To set up cross-account cross-Region functionality in your CloudWatch console, use the CloudWatch console to set up your sharing accounts and monitoring accounts.

Set up a sharing account

You must enable sharing in each account that will make data available to the monitoring account.

This will grant the read-only permissions that you choose in step 5 to all users that view a cross account dashboard in the account that you share with, if the user has corresponding permissions in the account that you share with.

To enable your account to share CloudWatch data with other accounts

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Settings**.

3. For **Share your CloudWatch data**, choose **Configure**.
4. For **Sharing**, choose **Specific accounts** and enter the IDs of the accounts that you want to share data with.

   Any accounts that you specify here can view your account's CloudWatch data. Specify the IDs only of accounts that you know and trust.
5. For **Permissions**, specify how to share your data with one of the following options:
   - **Provide read-only access to your CloudWatch metrics, dashboards, and alarms**. This option enables the monitoring accounts to create cross-account dashboards that include widgets that contain CloudWatch data from your account.
   - **Include CloudWatch automatic dashboards**. If you select this option, users in the monitoring account can also view the information in this account's automatic dashboards. For more information, see [Getting started with CloudWatch automatic dashboards](#).
   - **Include X-Ray read-only access for the X-Ray Trace Map**. If you select this option, users in the monitoring account can also view the X-Ray trace map and X-Ray trace information in this account. For more information, see [Using the X-Ray Trace Map](#).
   - **Full read-only access to everything in your account**. This option enables the accounts that you use for sharing to create cross-account dashboards that include widgets that contain CloudWatch data from your account. It also enables those accounts to look deeper into your account and view your account's data in the consoles of other Amazon services.
6. Choose **Launch CloudFormation template**.

   In the confirmation screen, type **`Confirm`**, and choose **Launch template**.
7. Select the **I acknowledge...** check box, and choose **Create stack**.

Sharing with an entire organization

Completing the preceding procedure creates an IAM role which enables your account to share data with one account. You can create or edit an IAM role that shares your data with all accounts in an organization. Do this only if you know and trust all accounts in the organization.

This will grant the read-only permissions listed in the policies shown in step 5 of the previous procedure to all users that view a cross-account dashboard in the account that you share with, if the user has corresponding permissions in the account that you share with.

To share your CloudWatch account data with all accounts in an organization

1. If you haven't already, complete the preceding procedure to share your data with one Amazon account.
2. Sign in to the Amazon Web Services Management Console and open the IAM console at https://console.amazonaws.cn/iam/.
3. In the navigation pane, choose **Roles**.
4. In the list of roles, choose **CloudWatch-CrossAccountSharingRole**.
5. Choose **Trust relationships**, **Edit trust relationship**.

   You see a policy like this:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::123456789012:root"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

6. Change the policy to the following, replacing *org-id* with the ID of your organization.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:PrincipalOrgID": "org-id"
        }
      }
```

```
    }
  ]
}
```

7. Choose **Update Trust Policy**.

Set up a monitoring account

Enable each monitoring account if you want to view cross-account CloudWatch data.

When you complete the following procedure, CloudWatch creates a service-linked role that CloudWatch uses in the monitoring account to access data shared from your other accounts. This service-linked role is called **AWSServiceRoleForCloudWatchCrossAccount**. For more information, see Using service-linked roles for CloudWatch.

To enable your account to view cross-account CloudWatch data

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Settings**, and then, in the **Cross-account cross-region** section, choose **Configure**.
3. Under the **View cross-account cross-region** section, choose **Enable**, and then select the **Show selector in the console** checkbox to enable an account selector to appear in the CloudWatch console when you're graphing a metric or creating an alarm.
4. Under **View cross-account cross-region**, choose one of the following options:

   - **Account Id Input**. This option prompts you to manually input an account ID each time that you want to switch accounts when you view cross-account data.
   - **Amazon Organization account selector**. This option causes the accounts that you specified when you completed your cross-account integration with Organizations to appear. When you next use the console, CloudWatch displays a dropdown list of these accounts for you to select from when you are viewing cross-account data.

     To do this, you must have first used your organization management account to allow CloudWatch to see a list of accounts in your organization. For more information, see (Optional) Integrate with Amazon Organizations.

- **Custom account selector**. This option prompts you to enter a list of account IDs. When you next use the console, CloudWatch displays a dropdown list of these accounts for you to select from when you are viewing cross-account data.

  You can also enter a label for each of these accounts to help you identify them when choosing accounts to view.

  The account selector settings that a user makes here are retained only for that user, not for all other users in the monitoring account.

5. Choose **Enable**.

After you complete this setup, you can create cross-account dashboards. For more information, see [Creating a customized CloudWatch dashboard](#).

Cross-Region functionality

Cross-Region functionality is built in to this feature automatically. You do not need to take any extra steps to be able to display metrics from different Regions in a single account on the same graph or the same dashboard. Cross-Region functionality is not supported for alarms, so you can't create an alarm in one Region that watches a metric in a different Region.

(Optional) Integrate with Amazon Organizations

If you want to integrate cross-account functionality with Amazon Organizations, you must make a list of all accounts in the organization available to the monitoring accounts.

To enable cross-account CloudWatch functionality to access a list of all accounts in your organization

1. Sign in to your organization's management account.
2. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
3. In the navigation pane, choose **Settings**, then choose **Configure**.
4. For **Grant permission to view the list of accounts in the organization**, choose **Specific accounts** to be prompted to enter a list of account IDs. The list of accounts in your organization are shared with only the accounts that you specify here.
5. Choose **Share organization account list**.
6. Choose **Launch CloudFormation template**.

In the confirmation screen, type **`Confirm`**, and choose **Launch template**.

Troubleshooting your CloudWatch cross-account setup

This section contains troubleshooting tips for cross-account, console deployment in CloudWatch.

I am getting access denied errors displaying cross-account data

Check the following:

- Your monitoring account should have a role named **AWSServiceRoleForCloudWatchCrossAccount**. If it does not, you need to create this role. For more information, see Set Up a Monitoring Account.
- Each sharing account should have a role named **CloudWatch-CrossAccountSharingRole**. If it does not, you need to create this role. For more information, see Set Up A Sharing Account.
- The sharing role must trust the monitoring account.

To confirm that your roles are set up properly for the CloudWatch cross-account console

1. Sign in to the Amazon Web Services Management Console and open the IAM console at https://console.amazonaws.cn/iam/.
2. In the navigation pane, choose **Roles**.
3. In the list of roles, make sure the needed role exists. In a sharing account, look for **CloudWatch-CrossAccountSharingRole**. In a monitoring account, look for **AWSServiceRoleForCloudWatchCrossAccount**.
4. If you are in a sharing account and **CloudWatch-CrossAccountSharingRole** already exists, choose **CloudWatch-CrossAccountSharingRole**.
5. Choose **Trust relationships**, **Edit trust relationship**.
6. Confirm that the policy lists either the account ID of the monitoring account, or the organization ID of an organization that contains the monitoring account.

I don't see an account dropdown in the console

First, check that you have created the correct IAM roles, as discussed in the preceding troubleshooting section. If those are set up correctly, make sure that you have enabled this account to view cross-account data, as described in Enable Your Account to View Cross-Account Data.

Disabling and cleaning up after using cross-account

To disable cross-account functionality for CloudWatch, follow these steps.

Step 1: Remove the cross-account stacks or roles

The best method is to remove the Amazon CloudFormation stacks that were used to enable cross-account functionality.

- In each of the sharing accounts, remove the **CloudWatch-CrossAccountSharingRole** stack.
- If you used Amazon Organizations to enable cross-account functionality with all accounts in an organization, remove the **CloudWatch-CrossAccountListAccountsRole** stack in the organization's management account.

If you didn't use the Amazon CloudFormation stacks to enable cross-account functionality, do the following:

- In each of the sharing accounts, delete the **CloudWatch-CrossAccountSharingRole** IAM role.
- If you used Amazon Organizations to enable cross-account functionality with all accounts in an organization, delete the **CloudWatch-CrossAccountSharing-ListAccountsRole** IAM role in the organization's management account.

Step 2: Remove the service-linked role

In the monitoring account, delete the **AWSServiceRoleForCloudWatchCrossAccount** service-linked IAM role.

Explore related telemetry

Computer systems can generate a large amount of telemetry, including both metrics and logs, and complex systems even more so. When looking at a specific set of telemetry, it can be a challenge to find other telemetry related to your initial set. It can take advanced training to gain the skills needed to find issues and troubleshoot them. Because systems are complex, understanding what is going on can involve viewing metrics and logs from many different services and resources, requiring context switching and navigating between systems.

The Amazon CloudWatch **Explore related** feature offers access to Amazon resource relationships, related metrics, and logs across service consoles, enhancing observability and efficiency for operators of all skill levels. When viewing alarms or anomalies in CloudWatch dashboards, or metrics in Amazon, users can quickly find and view metrics and logs for related resources in your system.

CloudWatch provides visibility into metrics and logs tied to specific resources, and the **Explore related** pane extends that by allowing you to correlate your infrastructure resources to your workloads with all of their associated telemetry. This gives you quick access to the information you need to troubleshoot infrastructure-related issues. You view the relationship between resources, and their related telemetry in the **Explore related** pane. The **Explore related** pane is accessed from CloudWatch or from other Amazon consoles showing resources or telemetry.

> **Note**
>
> **Explore related** is currently limited in accounts set up as monitoring accounts in CloudWatch cross-account observability. You should access **Explore related** from the source accounts where the resources are originally created and managed. In source accounts, you can navigate between connected resources, and view related logs and metrics.

The following topics discuss the details of exploring related telemetry.

Topics

What is related telemetry?

Related telemetry is metrics and log data from resources that are related to the current resource or service. Traditionally, you might look at the metrics and logs that are related to a single load balancer, or all telemetry related to Amazon EC2. The **Explore related** feature in Amazon CloudWatch adds an interactive *topology map*. The map is a resource-centric view where you can find metrics and logs for a specific resource, but you can also see how that resource is connected to other resources.

For example, if you are looking at the telemetry for a load balancer in the **Explore related** pane, besides the metrics and logs associated with that load balancer, the maps shows you the target groups for that load balancer. Selecting one of the target groups will then show you the Amazon EC2 instances associated with that target group. At each step in this process, the telemetry, including metrics and logs, for the selected resources are shown, making it easy to quickly find the telemetry you are looking for, or to explore the telemetry, looking for the cause of a specific issue.

How to access the Explore related pane

Within the CloudWatch console, there are multiple ways to access telemetry related to your current view. For example, if you are looking at a graph on a dashboard, and you want to view telemetry related to that graph or an aspect of the graph, you can choose to explore related data directly from that graph. From many places in the console, you can choose an **Explore related** menu item, or select a compass icon ( ) to show the **Explore related** pane.

You can access the exploring experience from entry points throughout the CloudWatch console (and other Amazon consoles), including:

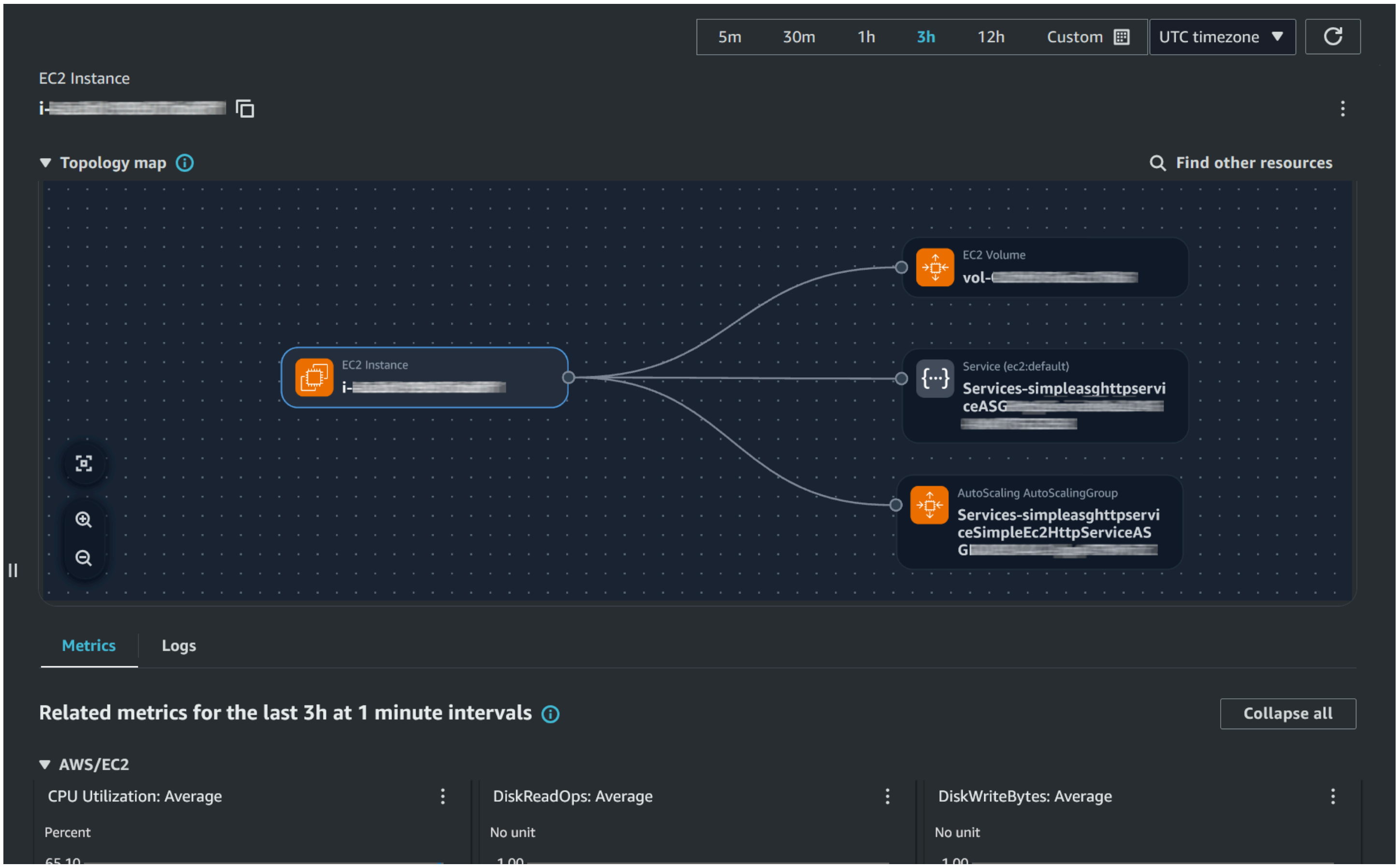
- **Metrics navigation –** When you choose **Metrics** then **All metrics** from the CloudWatch console left hand menu, the tile for any supported service or metric source will show the compass icon that brings up related telemetry in the bottom right corner.
- **Metric legend –** When viewing any metric graph (in CloudWatch or other Amazon consoles), hovering over the legend of the graph shows information about the data, as well as an **Explore related** button that brings up the related telemetry.
- **Metric datapoints –** When viewing any metric graph, hovering over a datapoint in the graph shows information about the metric, as well as a compass icon, to bring up the related telemetry.
- **Metric search –** When searching for metrics in CloudWatch, if you choose a found metric name, you can select **Explore related** from the menu that pops up, which will bring up the related telemetry.
- **Console toolbar –** In many Amazon console pages, the console toolbar (usually in the upper right of the console) includes a CloudWatch service icon, which will bring up CloudWatch tools, including the **Explore related** pane.

Depending on where you access the pane from, the default context of the pane will show an appropriate filter, if possible.

Navigating related telemetry

When you select any of the entry points to the **Explore related** pane, it appears on the right side of the CloudWatch console. This pane gives you access to view and find telemetry related to *entities*, in your system. An entity is a resource, such as an Amazon EC2 instance, or a service, such as an application that you have built. You can work within this pane without interrupting your current workflow, because it opens to the side of your initial page.

The following image shows the **Explore related** pane focused on a single Amazon EC2 instance, and the associated entities.



The top of the **Explore related** pane is a visual *topology map* (map) of the current entity and other related entities. The currently selected entity sets the focus for the pane. There are two ways to select an entity.

- Topology map – The *map* is a visual display of the current entity with focus. It also displays any related entities, allowing you to navigate around the set of resources and services that are related to each other.
- Find other resources – You can use the **Find other resources** button to filter and search for entities to use as the focus.

The bottom part of the pane shows you an automated search of metrics and logs for the current focal entity.

By default, the focus is set to an entity matching the location from which you accessed the **Explore related** pane. For example, if you accessed it by clicking the compass icon associated with a metric from a specific Amazon EC2 instance, then the focus will be set to that Amazon EC2 instance.

When you select an Amazon resource for the focus in the **Explore related** pane, you can navigate to the resource-specific console for the selected resource. For example, if you have selected an Amazon EC2 instance, you can select the **View in EC2 console** link to open the Amazon EC2 console with that instance selected.

When you set a focus, the metrics and logs are automatically filtered to show the telemetry related to your focus.

- **Metrics –** Each metric is shown as a graph for the time period that you have chosen.

  Much like any dashboard graph in CloudWatch, you can hover over or select a graph to get more information about the metric graph, and to see options including viewing in CloudWatch metrics. Choosing to view in CloudWatch will open the metrics view with the same viewing context as the **Explore related** pane, including resource and time range.
- **Log patterns –** CloudWatch analyzes the log groups associated with the focal resource and shows common patterns in those logs. For more information about log patterns, see Pattern analysis in the *Amazon CloudWatch Logs User Guide*.

  You can select **Compare timerange** to choose another time range and compare logs across the two time ranges.

  You can select **View in Logs Insights** to analyze the logs in CloudWatch Logs Insights with the same options as your current view, including resource, log group, and time range. For more information, see Analyzing log data with CloudWatch Logs Insights in the *Amazon CloudWatch Logs User Guide*.
- **Log groups –** The log groups that contain the logs are shown.

  You can select the log groups and then perform one of the following actions:

  - Choose **Start tailing in Live Tail** to view a streaming list of new log events as they are ingested for the selected log groups. The Live Tail session is started in the CloudWatch console. For more information on Live Tail, see Troubleshoot with CloudWatch Logs LiveTail in the *Amazon CloudWatch Logs User Guide*.
  - Choose **Query in Logs Insights** to open Logs Insights with a query scoped to just those log groups, applying your current context, including resources and time range.

Using the topology map

The *topology map* (map) is a visual display of the current focal entity and its related resources or services. You can use this interactive visualization to see the connections between different resources and services, and explore the relationships between components in your system. For example, if you are viewing *load balancer* resource, the map will show connected *target group* resources. Selecting a target group will display the associated instances. The visualization of connectivity helps operators understand and explore the relationships between different resources and services in your system.

You can drag the map, and zoom in and out, to see more of the associated entities, or to focus on fewer entities.

When you select an associated entity, like a target group, the pane's focus shifts to show telemetry for that entity. The map updates to center on the selected target group, displaying its connections to other entities, such as the load balancer and and any Amazon EC2 instances that are specified in that target group. As you navigate through different entities in the map, the metrics and logs at the bottom of the pane dynamically update, providing you with relevant telemetry for the newly selected resources.

Finding a specific resource

If a resource doesn't appear on the topology map, you can use the **Find other resources** feature to locate it. You can filter resources by tag or type, then select the ones you are looking for. After you have found resources to focus on, you are returned to the topology map, with those resources selected, to browse associated entities and telemetry.

> **Note**
>
> There are many reasons why you might not see your resources on the topology map. For example:
>
> - It's not related to the current focal entity.
> - You don't have permissions to access associated entities or telemetry.
> - The resource or service may not support telemetry or associated entities.

By using **Find other resources**, you can discover and visualize resources that may not be directly connected or visible in the current map. This ensures that you can access and analyze all relevant components of your infrastructure.

Select a resource with Find resources.

1. Open the **Explore related** pane from any of the entry points in the CloudWatch console.
2. Choose **Find resources**.
3. Choose the time frame for which you want to view logs or metrics.
4. Choose **Resource types**, then select the type of the resource you want to focus on from the drop down list, for example, **EC2 instances**.
5. Optionally, filter the set of resources by providing a tag to filter on. You can do this by selecting the **Filter resources by tags** filter, or by choosing the label that says **5 tags found** (the number will depend on the tags in your system). This gives you a list of tags to choose from.

   After you have selected the tags, the list of resources is automatically filtered to only those that are associated with those tags.
6. Optionally, choose one or more specific resources from the ones found that match your filters.
7. Choose **Show on map** to return to the **Topology map** with your resources selected.

Your list of **Metrics** and **Logs** is now filtered to just logs and metrics that are associated with that resource type. You can choose the **Metrics** or **Logs** tab to view the kind of telemetry that you want to view.

Permissions and prerequisites needed to view and explore related telemetry

To explore related telemetry, you must be getting entity information with the telemetry from your workloads, and you must have the proper permissions to view that data.

Many services send entity information automatically. For workloads that use the CloudWatch agent, you must have at least version `1.300049.1` of the agent, and you must configure it correctly. For information about configuring the agent, see How to add related information to custom telemetry sent to CloudWatch. For workloads running on Amazon EKS, you must have at least version `v2.3.1-eksbuild.1` of the Amazon CloudWatch Observability EKS add-on. For more information about this add-on, see Quick start with the Amazon CloudWatch Observability EKS add-on.

To explore related telemetry you must be signed in with certain permissions. Exploring related telemetry is a read-only activity, and requires at least read-only access to CloudWatch.

The permissions needed for viewing associations between telemetry and entities are: `logs:ListLogGroupsForEntity`, `logs:ListEntitiesForLogGroup`, `cloudwatch:ListEntitiesForMetric`, and `application-signals:ListObservedEntities`.

Each of the following Amazon managed policies will provide the CloudWatch permissions needed to access related telemetry in the CloudWatch console:

- CloudWatchFullAccessV2 – Provides full access to CloudWatch.
- CloudWatchReadOnlyAccess – Provides read-only access to CloudWatch.
- ReadOnlyAccess – Provides read-only access to Amazon services and resources.

Additionally, you must have at least read-only access (`Describe*` and `Get*`) to any resources in the topology map, in order for CloudWatch to discover and display relationships.

For more details about using policies to control access, see Managing access using policies.

How does CloudWatch find related telemetry?

The CloudWatch **Explore related** pane shows you metrics and logs that are related to each other, but how does that work?

Metrics and logs that are sent to CloudWatch can include an optional *entity* to which they are related. Typically, the entity will be a representation of what the telemetry is about. For example, a metric about CPU usage is about an Amazon EC2 instance, and will use that instance as its entity. When you view that metric in the **Explore related** pane, it shows you other telemetry for that same instance.

The *topology map* (map) in the **Explore related** pane displays the currently selected resource, along with *related resources*. For Amazon resources, CloudWatch automatically displays other resources that it knows are related. For example, if you are viewing an Amazon EC2 instance, the map will also display any Amazon EBS volumes that are attached to the instance. Selecting a volume shows telemetry for the volume, and the map is updated to display resources related to the volume. It also displays resources that are part of the same *service*.

The entity information associated with your telemetry defines the resource that the telemetry is associated with, such as the Amazon EC2 instance. However, it can also include contextual data about the resource. For example, if you have a website application that includes resources such as an Amazon EC2 instance and a database, the entity information can include the website application as a service. In this case, the topology map shows the service as a related entity, and when you select it, it displays the instances and database. This can make finding all the telemetry for a service simpler.

> **Note**
>
> CloudWatch must have received telemetry with entity information within the last three hours in order to find related resources and telemetry.

Where does the entity data come from?

There are different ways that CloudWatch gets entities for telemetry:

- Most telemetry sent to CloudWatch from Amazon services are associated with resources automatically. For a complete list of supported resources, see Amazon services that support related telemetry.
- The CloudWatch agent automatically adds entity information to the telemetry that it sends to CloudWatch.

  > **Note**
  >
  > You may need to update your CloudWatch agent to the latest version to include entity data. For more information, see Collect metrics, logs, and traces with the CloudWatch agent, and Configure CloudWatch agent service and environment names for related entities.

- When you are submitting your own telemetry, you can add entity information to the data. For more information, see How to add related information to custom telemetry sent to CloudWatch.

- CloudWatch makes a best effort to recognize the entity information associated with other telemetry (for example, custom telemetry that you send to CloudWatch without any entity information).

Where does service data come from?

Besides recognizing the natural connections between resources, such as an instance resource and an attached volume resource, CloudWatch can also group resources by *service*. For example, a service might be a website application. An Amazon EC2 instance with a web server, and another with a database might both be part of the same service, and are connected on the topology map based on that service.

There are different ways that CloudWatch gets a service name for telemetry, including:

- Application signals or otel instrumented telemetry use the `OTEL_SERVICE_NAME` environment variable used by supported OpenTelemetry instrumentation libraries to set the service name.
- The CloudWatch agent configuration allows configuring a service name. For more information, see Configure CloudWatch agent service and environment names for related entities.
- Kubernetes workloads use a corresponding name from the cluster, such as the Deployment, ReplicaSet, Pod, or Container, for the service name.
- For Amazon EC2 workloads, the service can come from tags (the `service`, `application`, or `app` tags).

> **Note**
>
> To use tags to generate service names, you must first set up instance metadata for the Amazon EC2 instance.

- When you are submitting your own telemetry, you can add service information to the data. For more information, see How to add related information to custom telemetry sent to CloudWatch.
- When it cannot use the above, CloudWatch uses the name of the IAM role that sends the metrics as the service name. This, for example, can provide a service name for Amazon ECS telemetry.

Amazon services that support related telemetry

The following table lists the Amazon services that support related entity information in their CloudWatch telemetry. Services or resources that are not listed in the table do not have related entity information exposed in CloudWatch.

> **Note**
>
> For services that use the CloudWatch agent, you may need to update the agent to the latest version to get related telemetry. For information about sending related entity information with your own custom metrics, see How to add related information to custom telemetry sent to CloudWatch.

Amazon Service	Resource	Metrics	Logs
Amazon API Gateway	`AWS::ApiGateway::Method`	Yes	No
Amazon API Gateway	`AWS::ApiGateway::Resource`	Yes	No
Amazon API Gateway	`AWS::ApiGateway::RestApi`	Yes	No
Amazon API Gateway	`AWS::ApiGateway::Stage`	Yes	Yes
Amazon API Gateway	`AWS::ApiGateway::VpcLink`	Yes	No
Amazon API Gateway V2	`AWS::ApiGatewayV2::Integration`	Yes	No

Amazon Service	Resource	Metrics	Logs
Amazon API Gateway V2	`AWS::ApiGatewayV2::Route`	Yes	No
Amazon API Gateway V2	`AWS::ApiGatewayV2::Stage`	No	Yes
Amazon API Gateway V2	`AWS::ApiGatewayV2::Api`	Yes	No
Amazon App Runner	`AWS::AppRunner::Service`	No	Yes
Amazon Application Migration Service	`AWS::MGN::SourceServer`	Yes	No
Amazon AppStream 2.0	`AWS::AppStream::Fleet`	Yes	Yes
Amazon AppSync	`AWS::AppSync::GraphQLApi`	Yes	Yes
Amazon B2B Data Interchange	`AWS::B2BI::Transformer`	No	Yes
Amazon Backup gateway	`AWS::BackupGateway::Hypervisor`	No	Yes
Amazon Bedrock	`AWS::Bedrock::KnowledgeBase`	No	Yes
Amazon Bedrock	`AWS::Bedrock::ModelId`	Yes	No

Amazon Service	Resource	Metrics	Logs
Amazon Chime	`AWS::Chime::SipMediaApplication`	No	Yes
Amazon Chime	`AWS::Chime::VoiceConnector`	No	Yes
Amazon Clean Rooms	`AWS::CleanRooms::Membership`	No	Yes
Amazon CloudFormation		No	Yes
Amazon CloudFormation Hooks		No	Yes
Amazon CloudFront	`AWS::CloudFront::Distribution`	Yes	Yes
Amazon CloudTrail	`AWS::CloudTrail::EventDataStore`	No	Yes
Amazon CloudTrail	`AWS::CloudTrail::Trail`	No	Yes
Amazon CloudWatch	`AWS::CloudWatch::MetricStream`	Yes	No
Amazon CloudWatch Evidently	`AWS::Evidently::Project`	No	Yes

Amazon Service	Resource	Metrics	Logs
Amazon CloudWatch Logs	AWS::Logs::LogGroup	Yes	No
Amazon CloudWatch RUM	AWS::RUM::AppMonitor	No	Yes
Amazon CloudWatch Synthetics	AWS::Synthetics::Canary	Yes	No
Amazon CodeBuild	AWS::CodeBuild::Project	Yes	No
Amazon CodeWhisperer	AWS::CodeWhisperer::Customization	No	Yes
Amazon Cognito user pools	AWS::Cognito::UserPool	Yes	Yes
Amazon Config	AWS::Config::ConfigRule	No	Yes
Amazon Connect	AWS::Connect::Instance	No	Yes
Amazon Database Migration Service	AWS::DMS::ReplicationInstance	Yes	No
Amazon Database Migration Service	AWS::DMS::ReplicationTask	Yes	No
Amazon DataSync	AWS::Datasync::Agent	Yes	No

Amazon Service	Resource	Metrics	Logs
Amazon DataSync	AWS::DataSync::Task	Yes	Yes
Amazon Directory Service	AWS::DirectoryService::MicrosoftAD	No	Yes
Amazon DynamoDB	AWS::DynamoDB::Table	Yes	No
DynamoDB Accelerator	AWS::DAX::Cluster	Yes	No
Amazon EC2	AWS::EC2::CapacityReservation	Yes	No
Amazon EC2	AWS::EC2::Instance	Yes	No
Amazon EC2	AWS::EC2::FlowLog	Yes	No
Amazon EC2	AWS::EC2::NATGateway	Yes	No
Amazon EC2	AWS::EC2::NetworkInterface	Yes	Yes
Amazon EC2	AWS::EC2::Subnet	Yes	Yes
Amazon EC2	AWS::EC2::TransitGateway	Yes	No

Amazon Service	Resource	Metrics	Logs
Amazon EC2	`AWS::EC2::TransitGatewayAttachment`	Yes	Yes
Amazon EC2	`AWS::EC2::VerifiedAccessInstance`	No	Yes
Amazon EC2	`AWS::EC2::Volume`	Yes	No
Amazon EC2	`AWS::EC2::VPC`	No	Yes
Amazon EC2	`AWS::EC2::VPNConnection`	Yes	Yes
Amazon EC2 Auto Scaling	`AWS::AutoScaling::AutoScalingGroup`	Yes	No
Amazon Elastic Beanstalk	`AWS::ElasticBeanstalk::Environment`	Yes	No
Amazon Elastic Container Service	`AWS::ECS::Cluster`	Yes	Yes
Amazon Elastic Container Service	`AWS::ECS::Service`	Yes	Yes
Amazon Elastic File System	`AWS::EFS::AccessPoint`	Yes	No

Amazon Service	Resource	Metrics	Logs
Amazon Elastic File System	AWS::EFS::FileSystem	Yes	No
Amazon Elastic File System	AWS::EFS::MountTarget	Yes	No
Amazon Elastic Kubernetes Service	AWS::EKS::Cluster	Yes	Yes
Amazon Elastic Kubernetes Service on Amazon Fargate		No	Yes
Elastic Load Balancing	AWS::ElasticLoadBalancing::LoadBalancer	Yes	No
Elastic Load Balancing V2	AWS::ElasticLoadBalancingV2::LoadBalancer	Yes	No
Elastic Load Balancing V2	AWS::ElasticLoadBalancingV2::TargetGroup	Yes	No
Amazon ElastiCache	AWS::ElastiCache::CacheCluster	Yes	Yes
AWS Elemental MediaConvert	AWS::MediaConvert::Queue	Yes	No

Amazon Service	Resource	Metrics	Logs
AWS Elemental MediaLive		No	Yes
AWS Elemental MediaLive	`AWS::MediaLive::Channel`	Yes	No
AWS Elemental MediaPackage	`AWS::MediaPackage::Channel`	Yes	No
AWS Elemental MediaStore	`AWS::MediaStore::Container`	Yes	Yes
AWS Elemental MediaTailor		No	Yes
Amazon EMR	`AWS::EMR::Cluster`	Yes	Yes
Amazon EventBridge	`AWS::Events::Rule`	Yes	Yes
Amazon EventBridge Pipes	`AWS::Pipes::Pipe`	Yes	Yes
Amazon Fault Injection Service	`AWS::FIS::ExperimentTemplate`	No	Yes
Amazon FinSpace	`AWS::FinSpace::Environment`	No	Yes
Amazon FSx	`AWS::FSx::FileSystem`	Yes	No

Amazon Service	Resource	Metrics	Logs
Amazon GameLift Servers	AWS::GameLift::Fleet	Yes	No
Amazon Glue	AWS::Glue::Job	No	Yes
Amazon Identity Sync	AWS::IdentitySync::Profile	No	Yes
Amazon Interactive Video Service	AWS::IVSChat::LoggingConfiguration	Yes	Yes
Amazon IoT	AWS::IoT::TopicRule	Yes	Yes
Amazon IoT 1-Click	AWS::IoT1Click::Device	Yes	No
Amazon IoT Analytics		No	Yes
Amazon IoT Events		No	Yes
Amazon IoT FleetWise	AWS::IoTFleetWise::Vehicle	No	Yes
Amazon IoT SiteWise		No	Yes
Amazon Key Management Service	AWS::KMS::Key	Yes	No
Amazon Managed Service for Apache Flink	AWS::KinesisAnalytics::Application	Yes	Yes

Amazon Service	Resource	Metrics	Logs
Amazon Data Firehose	AWS::KinesisFirehose::DeliveryStream	Yes	Yes
Amazon Kinesis Data Streams	AWS::Kinesis::Stream	Yes	No
Amazon Kinesis Video Streams	AWS::KinesisVideo::Stream	Yes	No
Amazon Lambda	AWS::Lambda::Function	Yes	No
Amazon Lex		No	Yes
Amazon Mainframe Modernization	AWS::M2::Application	No	Yes
Amazon Managed Streaming for Apache Kafka	AWS::Kafka::Cluster	Yes	No
Amazon Managed Streaming for Apache Kafka	AWS::KafkaConnect::Connector	No	Yes
Amazon Managed Streaming for Apache Kafka	AWS::MSK::Cluster	Yes	Yes
Amazon MemoryDB	AWS::Mmemorydb::Cluster	Yes	No
Amazon MQ	AWS::AmazonMQ::Broker	Yes	Yes

Amazon Service	Resource	Metrics	Logs
Amazon Network Firewall	`AWS::NetworkFirewall::Firewall`	Yes	Yes
Amazon OpenSearch Service	`AWS::OpenSearchService::Domain`	Yes	No
Amazon OpenSearch Service		No	Yes
Amazon OpenSearch Service Ingestion	`AWS::OSIS::Pipeline`	No	Yes
Amazon OpsWorks	`AWS::OpsWorks::Instance`	Yes	No
Amazon OpsWorks	`AWS::OpsWorks::Layer`	Yes	No
Amazon OpsWorks	`AWS::OpsWorks::Stack`	Yes	No
Amazon Organizations	`AWS::Organizations::Organization`	No	Yes
Amazon Outposts	`AWS::Outposts::Outpost`	Yes	No
Amazon Managed Service for Prometheus	`AWS::Prometheus::Resource`	Yes	No
Amazon Q Business		No	Yes

Amazon Service	Resource	Metrics	Logs
Amazon QLDB	AWS::QLDB ::Ledger	Yes	No
Amazon QuickSight	AWS::Quic ksight::D ashboard	Yes	No
Amazon QuickSight	AWS::Quic ksight::D ataSet	Yes	No
Amazon Redshift	AWS::Reds hift::Cluster	Yes	Yes
Amazon Redshift Serverless	AWS::Reds hiftServe rless::Wo rkgroup	Yes	No
Amazon Relational Database Service	AWS::RDS: :DBCluster	Yes	Yes
Amazon Relational Database Service	AWS::RDS: :DBInstance	Yes	Yes
Amazon RoboMaker	AWS::Robo Maker::Si mulationJob	Yes	No
Amazon Route 53	AWS::Rout e53::Heal thCheck	Yes	Yes
Amazon Route 53	AWS::Rout e53::Host edZone	Yes	No

Amazon Service	Resource	Metrics	Logs
Amazon Route 53 Resolver	`AWS::Route53Resolver::ResolverEndpoint`	Yes	No
Amazon S3	`AWS::S3::Bucket`	Yes	No
Amazon SageMaker AI	`AWS::SageMaker::Endpoint`	Yes	No
Amazon SageMaker AI	`AWS::SageMaker::Workteam`	No	Yes
Amazon Service Catalog	`AWS::ServiceCatalog::CloudFormationProduct`	Yes	No
Amazon Simple Email Service	`AWS::SES::ConfigurationSet`	Yes	No
Amazon Simple Notification Service	`AWS::SNS::Topic`	Yes	Yes
Amazon Simple Notification Service		No	Yes
Amazon Simple Queue Service	`AWS::SQS::Queue`	Yes	No
Amazon Step Functions	`AWS::StepFunctions::Activity`	Yes	No

Amazon Service	Resource	Metrics	Logs
Amazon Step Functions	AWS::StepFunctions::StateMachine	Yes	Yes
Amazon Storage Gateway	AWS::StorageGateway::Gateway	Yes	No
Amazon Storage Gateway	AWS::StorageGateway::Share	No	Yes
Amazon Transfer Family	AWS::Transfer::Server	Yes	Yes
Amazon VPC Lattice	AWS::VpcLattice::Service	No	Yes
Amazon WAFV2	AWS::WAFv2::WebAcl	No	Yes
Amazon WorkMail	AWS::WorkMail::Organization	Yes	No
Amazon WorkSpaces	AWS::WorkSpaces::Workspace	Yes	Yes

How to add related information to custom telemetry sent to CloudWatch

When you publish your own metrics and logs to CloudWatch, the entity information needed for related telemetry is not there by default. When you send metrics to CloudWatch or logs to CloudWatch Logs (with the PutMetricData or PutLogEvents operations), you can add entity information to those logs or metrics. The entity information is associated with the telemetry, and used in the **Explore related** feature to find related telemetry that is associated with the same entity.

The entity sent with telemetry represents a resource or service that the telemetry is associated with. For example, a metric about a service, or that comes from an Amazon resource. To identify the entity associated in your code, you create a set of `KeyAttributes` and optional `Attributes` of the entity.

> **Note**
>
> CloudWatch can only find related resources for entities that have had telemetry sent within the previous three hours. If your resource only emits sparse telemetry (less than once every 3 hours), you may want to send additional *heartbeat* telemetry, to keep the entity active within CloudWatch.

The following sections describe how to create the `KeyAttributes` and `Attributes` so that CloudWatch can identify the resources and services associated with the telemetry.

Identifying the entity with the KeyAttributes object

The `KeyAttributes` property of the Entity objects (in CloudWatch or in CloudWatch logs) uniquely identifies the entity for CloudWatch. It is a list of key-value pairs. Entities with the same `KeyAttributes` are considered to be the same entity. Telemetry associated with the same entity are considered related, and can be easily found in the **Explore related** pane.

> **Note**
>
> In the CloudWatch API, the property is called `KeyAttributes`. In the CloudWatch Logs API, the property is called `keyAttributes`. Here they are treated as the same property.

There are five possible types of object that an `Entity` can represent.

- **AWS::Resource** – The entity represents an Amazon resource, such as a DynamoDB table or Amazon EC2 instance.

- **AWS::Service –** The entity represent an Amazon service, such as Amazon S3. This might be used, for example, when calling the `ListBuckets` operation, which is not associated with a specific Amazon S3 resource.
- **Service –** The entity represents a workload running in your account. For example, an application or service that you manage.
- **Resource –** The entity represents a resource that is not managed by Amazon, for example, operating system resources, such as processes or file volumes.
- **RemoteService –** The entity represents an external service in a remote call. For example, a remote call to a database, external cache, or an external endpoint.

Depending on which of the above types you are trying to represent, you must provide the correct key-value pairs for the `KeyAttributes`. The following describes each type.

AWS::Resource

To specify an Amazon Resource, you must include the following three key value pairs:

- `"Type": "AWS::Resource"` – This key-value pair identifies the entity as an Amazon resource.
- `"ResourceType": "<resource-type>"` – The string value of the `ResourceType` is the Amazon CloudFormation resource type string. For example, `AWS::DynamoDB::Table`.
- `"Identifier": "<resource-id>"` – The primary identifier for the resource. For more information, see primaryIdentifier in the *Extension Development for CloudFormation User Guide*.

AWS::Service

To specify an Amazon Service, you must include the following two key value pairs:

- `"Type": "AWS::Service"` – This key-value pair identifies the entity as an Amazon service.
- `"Name": "<service-name>"` – The value of the `Name` is the Amazon CloudFormation service name string. For example, `AWS::DynamoDB`.

Service

To specify a service that is not operated by Amazon, you must include the following three key value pairs:

- `"Type": "Service"` – This key-value pair identifies the entity as a service.
- `"Name": "<service-name>"` – This represents the name of the service that is sending the metrics. For example, `my-service-frontend` or `api.myservice.com`.
- `"Environment": "<environment-name>"` – This attribute specifies where the service is hosted, or the environment to which it belongs. For example `us-west-2`, or `myservice.production`.

Resource

To specify a resource that is not provided by Amazon, you must include the following three key value pairs:

- `"Type": "Resource"` – This key-value pair identifies the entity as a resource.
- `"ResourceType": "<resource-type>"` – A string specifying the type of resource. For example, `K8s::Pod` for a Kubernetes Pod.
- `"Identifier": "<resource-id>"` – A string identifier for the resource. Can contain multiple names, separated by pipes. For example, a Kubernetes Pod might be represented by it's cluster name, namespace, and pod name, such as `MyCluster|MyNamespace|MyPod`.

RemoteService

To specify a remote service, you must include the following two key value pairs:

- `"Type": "RemoteService"` – This key-value pair identifies the entity as a remote service.
- `"Name": "<remote-service-name>"` – Specifies how the application refers to the external service in a remote call. For example, `api.test.myservice.com`.

Providing additional details about the entity with the Attributes object

You can provide additional details about the `Entity` that you provide with your telemetry. This can include details about the platform, resource, application, or telemetry provider. The following tables describes the key words that you can use for each of these types of data.

> **Note**
>
> In the CloudWatch API, the property is called `Attributes`. In the CloudWatch Logs API, the property is called `attributes`. Here they are treated as the same property.

Platform details

Keyword	Usage	Domain of Values	Examples
`PlatformType`	Defines the hosted-in platform.	AWS::EKS, AWS::ECS, AWS::EC2, AWS::Lambda, K8s, Generic	`AWS::EC2`
`EKS.Cluster`	Name of the Amazon EKS cluster.	Alphanumeric string with basic delimiters.	`FlyingSquad`
`K8s.Cluster`	Name of the self-hosted Kubernetes cluster.	Alphanumeric string with basic delimiters.	`minicube`
`K8s.Namespace`	Name of Kubernetes namespace in Amazon EKS or K8s clusters.	Alphanumeric string with basic delimiters.	`default`, `pet-clinic`
`K8s.Workload`	Name of Kubernetes workload in Amazon EKS and K8s clusters.	Alphanumeric string with basic delimiters.	`frontend`
`K8s.Node`	Identity of Kubernetes node in Amazon EKS and K8s clusters.	K8s node name (for example, Amazon EC2 instance DNS name).	`ip-11-22-33-44.ec2.internal`

Keyword	Usage	Domain of Values	Examples
`K8s.Pod`	Identity of Kubernetes pod in Amazon EKS and K8s clusters.	K8s pod identifier.	`frontend-1234abcd56-ef7890`
`EC2.AutoScalingGroup`	Name of the Amazon EC2 AutoScaling Group.	Alphanumeric string with basic delimiters.	`my-asg-name-1`
`EC2.InstanceId`	Identity of the Amazon EC2 instance.	Amazon EC2 instance identifier.	`i-1234abcd5678ef90`
`ECS.Cluster`	Identity of the Amazon ECS cluster.	Amazon ECS cluster name.	`MyCluster`
`ECS.Service`	Identity of the Amazon ECS service.	Amazon ECS service name.	`MyService`
`ECS.Task`	Identity of the Amazon ECS task.	Amazon ECS task ID.	`task-123abc`
`Lambda.Function`	Identity of the Lambda function.	Lambda function name.	`MyFunction`
`Host`	Name of the host for all platform types.	Sub-domain format.	`ip-111-22-33-44.example.com`

Resource details

Keyword	Usage	Domain of Values	Examples
`AWS.Resource.ARN`	ARN for the Amazon resource.	Alphanumeric string with basic delimiters.	`arn:aws:dynamodb:us-east-1:123456789`

Keyword	Usage	Domain of Values	Examples
			012:table/ myDynamoDBTable

Application details

Keyword	Usage	Domain of Values	Examples
AWS.Application	Name of the application in AppRegistry.	Alphanumeric string with basic delimiters.	PetClinicApp
AWS.Application.ARN	ARN of the application in AppRegistry.	Alphanumeric string with basic delimiters.	arn:aws:servicecatalog:us-east-1:1234567890:/applications/...

Telemetry provider details

Keyword	Usage	Domain of Values	Examples
Telemetry.SDK	The fingerprint of OTEL SDK version for instrumented services.	Alphanumeric string with basic delimiters.	opentelemetry,1.32.0-aws-SNAPSHOT,java,Auto
Telemetry.Agent	The fingerprint of the Agent used to collect and send telemetry data.	Alphanumeric string with basic delimiters.	CWAgent/1.300026.3, ADOTCollector/1.x
Telemetry.Source	Specifies the point of application where	ServerSpan, ClientSpan,	ClientSpan, JMX

Keyword	Usage	Domain of Values	Examples
	the telemetry was collected or what was used for the source of telemetry data.	ProducerSpan, ConsumerSpan, LocalRoot Span, JMX, OS.	

Query metrics from other data sources

You can use CloudWatch to query, visualize, and create alarms for metrics from other data sources. To do so, you connect CloudWatch to the other data sources. This gives you a single, consolidated monitoring experience within the CloudWatch console. You can have a unified view of your infrastructure and application metrics regardless of where the data is stored, helping you identify and resolve issues faster.

After you connect to a data source using a CloudWatch wizard, CloudWatch creates an Amazon CloudFormation stack that deploys and configures an Amazon Lambda function. This Lambda function runs on demand every time you query the data source. The CloudWatch query builder shows you in real time a list of elements that can be queried, such as metrics, tables, fields, or labels. As you make choices, the query builder pre-populates a query in the native language of the selected source.

CloudWatch provides guided wizards for you to connect to the following data sources. For these data sources, you provide basic information to identify the data source and credentials. You can also manually create connectors to other data sources by creating your own Lambda functions.

- Amazon OpenSearch Service– Derive metrics from your OpenSearch Service logs and traces.
- Amazon Managed Service for Prometheus– Query these metrics using PromQL.
- Amazon RDS for MySQL – Use SQL to convert data stored in your Amazon RDS tables into metrics.
- Amazon RDS for PostgreSQL– Use SQL to convert data stored in your Amazon RDS tables into metrics.
- Amazon S3 CSV files– Display metrics data from a CSV file stored in an Amazon S3 bucket.
- Microsoft Azure Monitor– Query metrics from your Microsoft Azure Monitor account.
- Prometheus– Query these metrics using PromQL.

After you create connectors to data sources, see Creating a graph of metrics from another data source for information about graphing a metric from a data source. For information about setting an alarm on a metric from a data source, see Create an alarm based on a connected data source.

Topics

- Connect to a prebuilt data source with a wizard
- Create a custom connector to a data source
- Use your custom data source
- Delete a connector to a data source

Managing access to data sources

CloudWatch uses Amazon CloudFormation to create the required resources in your account. We recommend that you use the `cloudformation:TemplateUrl` condition to control access to Amazon CloudFormation templates when you grant `CreateStack` permissions to IAM users.

Warning

Any user that you grant data source invoke permission to can query metrics from that data source even if that user does not have direct IAM permissions to the data source. For example, if you grant `lambda:InvokeFunction` permissions on a Amazon Managed Service for Prometheus data source Lambda function to a user, that user will be able to query metrics from the corresponding Amazon Managed Service for Prometheus workspace even if you didn't grant them direct IAM access to that workspace.

You can find template URLs for data sources on the **Create stack** page in the CloudWatch Settings Console.

```
{
  "Version":"2012-10-17",
  "Statement":[
    {
      "Effect" : "Allow",
      "Action" : [ "cloudformation:CreateStack" ],
      "Resource" : "*",
      "Condition" : {
        "StringEquals" : {
          "cloudformation:TemplateUrl" : [ data-source-template-url ]
        }
      }
    }
  ]
```

```
}
```

For more information about controlling Amazon CloudFormation access, see Controlling access with Amazon Identity and Access Management

Connect to a prebuilt data source with a wizard

This topic provides instructions for using the wizard to connect CloudWatch to the following data sources.

- Amazon OpenSearch Service
- Amazon Managed Service for Prometheus
- Amazon RDS for MySQL
- Amazon RDS for PostgreSQL
- Amazon S3 CSV files
- Microsoft Azure Monitor
- Prometheus

The subsections in this topic include notes about managing and querying with each of these data sources.

To create a connector to a data source

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Settings**.
3. Choose the **Metrics data sources** tab.
4. Choose **Create data source**.
5. Select the source that you want, then choose **Next**.
6. Enter a name for the data source.
7. Enter the other required information, depending on the data source that you chose. This can include credentials to access the data source and data source identifying information such as Prometheus workspace name, database name, or Amazon S3 bucket name. For Amazon services, the wizard discovers the resources and populates them into the selection dropdown.

   For more notes about the data source you are using, see the sections after this procedure.

8. To have CloudWatch connect to the data source in a VPC, choose **Use a VPC** and select the VPC to use. Then select the subnet and security group.
9. Choose **I acknowledge that Amazon CloudFormation will create IAM resources**. This resource is the Lambda function execution role.
10. Choose **Create data source**.

    The new source that you just added doesn't appear until the Amazon CloudFormation stack is done creating it. To check progress, you can choose **View the status of my CloudFormation stack**. Or you can choose the refresh icon to update this list.

    When your new data source appears in this list, it is ready to use. You can choose **Query from CloudWatch metrics** to begin querying with it. For more information, see Creating a graph of metrics from another data source.

Amazon Managed Service for Prometheus

Updating the data source configuration

- You can update your data source manually by doing the following:
  - To update the Amazon Managed Service for Prometheus workspace ID, update the `AMAZON_PROMETHEUS_WORKSPACE_ID` environment variable for the data source connector Lambda function.
  - To update the VPC configuration, see Configuring VPC access (console) for more information.

Querying the data source

- When querying Amazon Managed Service for Prometheus, after you select the data source in the **Multi source query** tab and select an Amazon Managed Service for Prometheus connector, you can use the **Query helper** to discover metrics and labels and provide simple PromQL queries. You can also use the PromQL query editor to build a PromQL query.
- Multi-line queries are not supported by the CloudWatch data source connectors. Every line feed is replaced with a space when the query is executed, or when you create an alarm or a dashboard widget with the query. In some cases, this might make your query not valid. For example, if your query contains a single line comment it will not be valid. If you try to create a dashboard or alarm with a multi-line query from the command line or Infrastructure as Code, the API will reject the action with a parse error.

Amazon OpenSearch Service

Creating the data source

If the OpenSearch domain is enabled for FGAC, you must map the execution role of the connector Lambda function to a user in OpenSearch Service. For more information, see the **Mapping users to roles** section in Managing permissions in the OpenSearch Service documentation.

If your OpenSearch domain is only accessible within a Virtual Private Cloud (VPC), you need to manually include a new environment variable in the Lambda function called `AMAZON_OPENSEARCH_ENDPOINT`. The value for this variable should be the root domain of the OpenSearch endpoint. You can obtain this root domain by removing `https://` and `<region>.es.amazonaws.com` from the domain endpoint listed in the OpenSearch Service console. For example, if your domain endpoint is `https://sample-domain.us-east-1.es.amazonaws.com`, the root domain would be `sample-domain`.

Updating the data source

- You can update your data source manually by doing the following:
  - To update the OpenSearch Service domain, update the `AMAZON_OPENSEARCH_DOMAIN_NAME` environment variable for the data source connector Lambda function.
  - To update the VPC configuration, see Configuring VPC access (console) for more information.

Querying the data source

- When querying OpenSearch Service, after you select the data source in the **Multi source query** tab, do the following:
  - Select the Index to query.
  - Select the Metric name (Any numeric field in the document) and Stat.
  - Select the Time axis (Any date field in the document).
  - Select Filters to apply (Any String field in the document).
  - Choose **Graph query**.

Amazon RDS for PostgreSQL and Amazon RDS for MySQL

Creating the data source

- If your data source is only accessible in a VPC, you must include the VPC configuration for the connector, as described in Connect to a prebuilt data source with a wizard. If the data source is to connect to the VPC for credentials, the endpoint must be configured in the VPC. For more information, see Using an Amazon Secrets Manager VPC endpoint.

  Additionally, you must create a VPC endpoint for the Amazon RDS service. For more information, see Amazon RDS API and interface VPC endpoints (Amazon PrivateLink).

Updating the data source

- You can update your data source manually by doing the following:
  - To update the database instance, update the `RDS_INSTANCE` environment variable for the data source connector Lambda function.
  - To update the username and password used to connect to Amazon RDS, use Amazon Secrets Manager. You can find the ARN of the secret used for the data source in the environment variable `RDS_SECRET` on the data source Lambda function. For more information about updating the secret in Amazon Secrets Manager, see Modify an Amazon Secrets Manager secret.
  - To update the VPC configuration, see Configuring VPC access (console) for more information.

Querying the data source

- When querying Amazon RDS, after you select the data source in the **Multi source query** tab and select an Amazon RDS connector, you can use the database discoverer to view available databases, tables, and columns. You can also use the SQL editor to create an SQL query.

  You can use the following variables in the query:
  - `$start.iso` – The start time in ISO date format
  - `$end.iso` – The end time in ISO date format
  - `$period` – The selected period in seconds

  For example, you can query `SELECT value, timestamp FROM table WHERE timestamp BETWEEN $start.iso and $end.iso`
- Multi-line queries are not supported by the CloudWatch data source connectors. Every line feed is replaced with a space when the query is executed, or when you create an alarm or a dashboard widget with the query. In some cases, this might make your query not valid. For example, if

your query contains a single line comment it will not be valid. If you try to create a dashboard or alarm with a multi-line query from the command line or Infrastructure as Code, the API will reject the action with a parse error.

> **Note**
>
> If no date field is found in the results, the values for each numeric field are summed to single values and plotted across the provided time range. If the timestamps don't align with the selected period in CloudWatch, the data is automatically aggregated using `SUM` and aligned with the period in CloudWatch.

Amazon S3 CSV files

Querying the data source

- When querying Amazon S3 CSV files, after you select the data source in the **Multi source query** tab and select an Amazon S3 connector, you select the Amazon S3 bucket and key.

  The CSV file must be formatted in the following ways:

  - The time stamp must be the first column.
  - The table must have a header row. The headers are used to name your metrics. The title of the time stamp column will be ignored, only the titles of the metrics columns are used.
  - The time stamps must be in ISO date format.
  - The metrics must be numeric fields.

  ```
  Timestamp, Metric-1, Metric-2, ...
  ```

  The following is an example:

timestamp	CPU (%)	Memory (%)	Storage (%)
2023-11-23T17:09:41+00:00	1	2	3

timestamp	CPU (%)	Memory (%)	Storage (%)
2023-11-23T17:04:41+00:00	4	5	6
2023-11-23T16:59:41+00:00	7	8	9
2023-11-23T16:54:41+00:00	10	11	12

Note

If no timestamp is provided, the values for each metric are summed to single values and plotted across the provided time range. If the timestamps don't align with the selected period in CloudWatch, the data is automatically aggregated using `SUM` and aligned with the period in CloudWatch.

Microsoft Azure Monitor

Creating the data source

- You must provide your tenant ID, client ID, and client secret to connect to Microsoft Azure Monitor. The credentials will be stored in Amazon Secrets Manager. For more information, see [Create a Microsoft Entra application and service principal that can access resources] in the Microsoft documentation.

Updating the data source

- You can update your data source manually by doing the following:
  - To update the tenant ID, client ID, and client secret used to connect to Azure Monitor, you can find the ARN of the secret used for the data source as the `AZURE_CLIENT_SECRET` environment variable on the data source Lambda function. For more information about updating the secret in Amazon Secrets Manager, see [Modify an Amazon Secrets Manager secret].

Querying the data source

- When querying Azure Monitor, after you select the data source in the **Multi source query** tab and select an Azure Monitor connector, you specify the Azure subscription, and the resource group and resource. You can then select the metric namespace, metric, and aggregation, and filter by dimensions.

Prometheus

Creating the data source

- You must provide the Prometheus endpoint and the user and password required to query Prometheus. The credentials will be stored in Amazon Secrets Manager.
- If your data source is only accessible in a VPC, you must include the VPC configuration for the connector, as described in Connect to a prebuilt data source with a wizard. If the data source is to connect to for credentials, the endpoint must be configured in the VPC. For more information, see Using an Amazon Secrets Manager VPC endpoint.

Updating data source configuration

- You can update your data source manually by doing the following:
  - To update the Prometheus endpoint, specify the new endpoint as the `PROMETHEUS_API_ENDPOINT` environment variable on the data source Lambda function.
  - To update the username and password used to connect to Prometheus, you can find the ARN of the secret used for the data source as the `PROMETHEUS_API_SECRET` environment variable on the data source Lambda function. For more information about updating the secret in Amazon Secrets Manager, see Modify an Amazon Secrets Manager secret.
  - To update the VPC configuration, see Configuring VPC access (console) for more information.

Querying the data source

> **Important**
>
> Prometheus metric types are different than CloudWatch metrics and many metrics available through Prometheus are cumulative by design. When you query Prometheus metrics, CloudWatch doesn't apply any additional transformation to the data: if you

specify only the metric name or label, the displayed value will be cumulative. For more information, see Metric types in the Prometheus documentation.

To see Prometheus metrics data as discrete values, like CloudWatch metrics, you need to edit the query before you run it. For example, you might need to add a call to the rate function over your Prometheus metric name. For documentation about the rate function and other Prometheus functions, see rate() in the Prometheus documentation.

Multi-line queries are not supported by the CloudWatch data source connectors. Every line feed is replaced with a space when the query is executed, or when you create an alarm or a dashboard widget with the query. In some cases, this might make your query not valid. For example, if your query contains a single line comment it will not be valid. If you try to create a dashboard or alarm with a multi-line query from the command line or Infrastructure as Code, the API will reject the action with a parse error.

Notification of Available Updates

From time to time, Amazon might notify you that we recommend that you update your connectors with a newer available version and will provide instructions for how to do so.

Create a custom connector to a data source

This topic describes how to connect a custom data source to CloudWatch. You can connect a custom data source to CloudWatch in two ways:

- Using a sample template that CloudWatch provides. You can use either JavaScript or Python with this template. These templates include sample Lambda code that will be useful to you when you create your Lambda function. You can then modify the Lambda function from the template to connect to your custom data source.
- Creating an Amazon Lambda function from scratch that implements the data source connector, the data query, and the preparation of the time series for use by CloudWatch. This function must pre-aggregate or merge datapoints if needed, and also align the period and timestamps to be compatible with CloudWatch.

Contents

Use a template

Using a template creates a sample Lambda function, and can help you get your custom connector built faster. These sample functions provide sample code for many common scenarios involved with building a custom connector. You can examine the Lambda code after you create a connector with a template, then modify it to use to connect to your data source.

Additionally, if you use the template, CloudWatch takes care of creating the Lambda permissions policy and attaching resource tags to the Lambda function.

To use the template to create a connector to a custom data source

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Settings**.
3. Choose the **Metrics data sources** tab.
4. Choose **Create data source**.
5. Choose the radio button for **Custom - getting started template** and then choose **Next**.
6. Enter a name for the data source.
7. Select the one of the listed templates.
8. Select either Node.js or Python.
9. Choose **Create data source**.

   The new custom source that you just added doesn't appear until the Amazon CloudFormation stack finishes creating it. To check progress, you can choose **View the status of my CloudFormation stack**. Or you can choose the refresh icon to update this list.

When your new data source appears in this list, it is ready for you to test in the console and modify.

10. (Optional) To query the test data from this source in the console, follow the instructions in Creating a graph of metrics from another data source.
11. Modify the Lambda function for your needs.

    a. In the navigation pane, choose **Settings**.
    b. Choose the **Metrics data sources** tab.
    c. Choose **View in Lambda console** for the source that you want to modify.

    You can now modify the function to access your data source. For more information, see Step 1: Create the function.

> **Note**
>
> By using the template, when you write your Lambda function you don't need to follow the instructions in Step 2: Create a Lambda permissions policy or Step 3: Attach a resource tag to the Lambda function. These steps were performed by CloudWatch because you used the template.

Create a custom data source from scratch

Follow the steps in this section to create a Lambda function that connects CloudWatch to a data source.

Step 1: Create the function

A custom data source connector must support `GetMetricData` events from CloudWatch. Optionally, you can also implement a `DescribeGetMetricData` event to provide documentation to users in the CloudWatch console for how to use the connector. The `DescribeGetMetricData` response can also be used to set defaults that are used in the CloudWatch custom query builder.

CloudWatch provides code snippets as samples to help you get started. For more information, see the samples repository at https://github.com/aws-samples/cloudwatch-data-source-samples.

Constraints

- The response from Lambda must be smaller than 6 Mb. If the response exceeds 6 Mb, the `GetMetricData` response marks the Lambda function as `InternalError` and no data is returned.
- The Lambda function must complete its execution within 10 seconds for visualization and dashboarding purposes, or within 4.5 seconds for alarms usage. If the execution time exceeds that time, the `GetMetricData` response marks the Lambda function as `InternalError` and no data is returned.
- The Lambda function must send its output using epoch timestamps in seconds.
- If the Lambda function doesn't resample the data and instead returns data that doesn't correspond to the start time and period length that was requested by the CloudWatch user, that data is ignored by CloudWatch. The extra data is discarded from any visualization or alarming. Any data that is not between the start time and end time is also discarded.

  For example, if a user asks for data from 10:00 to 11:00 with a period of 5 min, then "10:00:00 to 10:04:59" and "10:05:00 to 10:09:59" are the valid time ranges for data to be returned. You must return a time series that includes `10:00 value1`, `10:05 value2`, and so on. If the function returns `10:03 valueX`, for example, it gets dropped because 10:03 does not correspond to the requested start time and period.
- Multi-line queries are not supported by the CloudWatch data source connectors. Every line feed is replaced with a space when the query is executed, or when you create an alarm or a dashboard widget with the query. In some cases, this might make your query not valid.

GetMetricData event

Request payload

The following is an example of a `GetMetricData` request payload sent as input to the Lambda function.

```
{
  "EventType": "GetMetricData",
  "GetMetricDataRequest": {
    "StartTime": 1697060700,
    "EndTime": 1697061600,
    "Period": 300,
    "Arguments": ["serviceregistry_external_http_requests{host_cluster!=\"prod\"}"]
  }
}
```

- **StartTime–** The timestamp specifying the earliest data to return. The **Type** is timestamp epoch seconds.
- **EndTime–** The timestamp specifying the latest data to return. The **Type** is timestamp epoch seconds.
- **Period–** The number of seconds that each aggregation of the metrics data represents. The minimum is 60 seconds. The **Type** is Seconds.
- **Arguments–** An array of arguments to pass to the Lambda metric math expression. For more information about passing arguments, see [How to pass arguments to your Lambda function].

Response payload

The following is an example of a `GetMetricData` response payload returned by the Lambda function.

```
{
   "MetricDataResults": [
      {
         "StatusCode": "Complete",
         "Label": "CPUUtilization",
         "Timestamps": [ 1697060700, 1697061000, 1697061300 ],
         "Values": [ 15000, 14000, 16000 ]
      }
   ]
}
```

The response payload will contain either a `MetricDataResults` field or an `Error` field, but not both.

A `MetricDataResults` field is a list of time-series fields of type `MetricDataResult`. Each of those time-series fields can include the following fields.

- **StatusCode–** (Optional) `Complete` indicates that all data points in the requested time range were returned. `PartialData` means that an incomplete set of data points were returned. If this is omitted, the default is `Complete`.

  Valid Values: `Complete` | `InternalError` | `PartialData` | `Forbidden`
- **Messages–** Optional list of messages with additional information about the data returned.

  Type: Array of [MessageData] objects with `Code` and `Value` strings.

- **Label**– The human-readable label associated with the data.

  Type: String

- **Timestamps**– The timestamps for the data points, formatted in epoch time. The number of timestamps always matches the number of values and the value for `Timestamps[x]` is `Values[x]`.

  Type: Array of timestamps

- **Values**– The data point values for the metric, corresponding to `Timestamps`. The number of values always matches the number of timestamps and the value for `Timestamps[x]` is `Values[x]`.

  Type: Array of doubles

For more information about `Error` objects, see the following sections.

Error response formats

You can optionally use the error response to provide more information about errors. We recommend that you return an error with Code Validation when a validation error occurs, such as when a parameter is missing or is the wrong type.

The following is an example of the response when the Lambda function wants to raise a `GetMetricData` validation exception.

```
{
   "Error": {
      "Code": "Validation",
      "Value": "Invalid Prometheus cluster"
   }
}
```

The following is an example of the response when the Lambda function indicates that it's unable to return data because of an access issue. The response is translated into a single time series with a status code of `Forbidden`.

```
{
   "Error": {
      "Code": "Forbidden",
```

```
        "Value": "Unable to access ..."
    }
}
```

The following is an example of when the Lambda function raises an overall `InternalError` exception, which is translated into a single time series with a status code of `InternalError` and a message. Whenever an error code has a value other than `Validation` or `Forbidden`, CloudWatch assumes that it's a generic internal error.

```
{
   "Error": {
      "Code": "PrometheusClusterUnreachable",
      "Value": "Unable to communicate with the cluster"
   }
}
```

DescribeGetMetricData event

Request payload

The following is an example of a `DescribeGetMetricData` request payload.

```
{
  "EventType": "DescribeGetMetricData"
}
```

Response payload

The following is an example of a `DescribeGetMetricData` response payload.

```
{
    "Description": "Data source connector",
    "ArgumentDefaults": [{
        Value: "default value"
     }]
}
```

- **Description–** A description of how to use the data source connector. This description will appear in the CloudWatch console. Markdown is supported.

  Type: String

- **ArgumentDefaults**– Optional array of argument default values used pre-populate the custom data source builder.

  If `[{ Value: "default value 1"}, { Value: 10}]`, is returned, the query builder in the CloudWatch console displays two inputs, the first with “default value 1” and the second with 10.

  If `ArgumentDefaults` is not provided, a single input is displayed with type default set to `String`.

  Type: Array of objects containing Value and Type.
- **Error**– (Optional) An error field can be included in any response. You can see examples in GetMetricData event.

Important considerations for CloudWatch alarms

If you are going to use the data source to set CloudWatch alarms, you should set it up to report data with timestamps every minute to CloudWatch. For more information and other considerations for creating alarms on metrics from connected data sources, see Create an alarm based on a connected data source.

(Optional) Use Amazon Secrets Manager to store credentials

If your Lambda function needs to use credentials to access the data source, we recommend using Amazon Secrets Manager to store these credentials instead of hardcoding them into your Lambda function. For more information about using Amazon Secrets Manager with Lambda, see Use Amazon Secrets Manager secrets in Amazon Lambda functions.

(Optional) Connect to a data source in a VPC

If your data source is in a VPC managed by Amazon Virtual Private Cloud, you must configure your Lambda function to access it. For more information, see Connecting outbound networking to resources in a VPC.

You might also need to configure VPC service endpoints to access services such as Amazon Secrets Manager. For more information, see Access an Amazon service using an interface VPC endpoint.

Step 2: Create a Lambda permissions policy

You must use create a policy statement that grants CloudWatch permission to use the Lambda function that you created. You can use the Amazon CLI or the Lambda console to create the policy statement.

To use the Amazon CLI to create the policy statement

- Enter the following command. Replace *123456789012* with your account ID, replace *my-data-source-function* with the name of your Lambda function, and replace *MyDataSource-DataSourcePermission1234* with an arbitrary unique value.

```
aws lambda add-permission --function-name my-data-source-function --statement-id MyDataSource-DataSourcePermission1234 --action lambda:InvokeFunction --principal lambda.datasource.cloudwatch.amazonaws.com --source-account 123456789012
```

Step 3: Attach a resource tag to the Lambda function

The CloudWatch console determines which of your Lambda functions are data source connectors by using a tag. When you create a data source using one of the wizards, the tag is automatically applied by the Amazon CloudFormation stack that configures it. When you create a data source yourself, you can use the following tag for your Lambda function. This makes your connector appear in the **Data source** dropdown in the CloudWatch console when you query metrics.

- A tag with `cloudwatch:datasource` as the key and `custom` as the value.

Use your custom data source

After you create a data source, you can use it to query and visualize data from that source, as well as to set alarms. If you used a template to create your custom data source connector or you added the tag listed in Step 3: Attach a resource tag to the Lambda function, you can follow the steps in Creating a graph of metrics from another data source to query it. You can also use the metric math function `LAMBDA` to query it, as explained in the following section. For information about creating alarms on metrics from your data source, see Create an alarm based on a connected data source. This topic describes how to pass arguments to your Lambda function to your custom data source.

How to pass arguments to your Lambda function

The recommended way for you to pass arguments to your custom data source is to use the query builder in the CloudWatch console when you query the data source.

You can also use your Lambda function to retrieve data from your data source by using the new LAMBDA expression in CloudWatch metric math.

```
LAMBDA("LambdaFunctionName" [, optional-arg]*)
```

`optional-arg` is up to 20 strings, numbers, or Booleans. For example, `param`, `3.14`, or `true`.

> **Note**
>
> Multi-line strings are not supported by the CloudWatch data source connectors. Every line feed is replaced with a space when the query is executed, or when you create an alarm or a dashboard widget with the query. In some cases, this might make your query not valid.

When you use the LAMBDA metric math function, you can provide the function name (`"MyFunction"`). If your resource policy allows, you can also use a specific version of the function (`"MyFunction:22"`), or a Lambda function alias (`"MyFunction:MyAlias"`). You can't use *

The following are some examples of calling the LAMBDA function.

```
LAMBDA("AmazonOpenSearchDataSource", "MyDomain", "some-query")
```

```
LAMBDA("MyCustomDataSource", true, "fuzzy", 99.9)
```

The LAMBDA metric math function returns a list of time series that can be returned to the requester or combined with other metric math functions. The following is an example of combining LAMBDA with other metric math functions.

```
FILL(LAMBDA("AmazonOpenSearchDataSource", "MyDomain", "some-query"), 0)
```

Delete a connector to a data source

The instructions in this section describe how to delete a connector to a data source.

To delete a connector to a data source

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Settings**.
3. Choose the **Metrics data sources** tab.
4. Choose **Manage in CloudFormation** in the row of the data source that you want to delete.

   You are taken to the Amazon CloudFormation console.
5. In the section with the name of your data source, choose **Delete**.
6. In the confirmation pop-up, choose **Delete**.

Collect metrics, logs, and traces with the CloudWatch agent

The unified CloudWatch agent enables you to do the following:

- Collect internal system-level metrics from Amazon EC2 instances across operating systems. The metrics can include in-guest metrics, in addition to the metrics for EC2 instances. The additional metrics that can be collected are listed in Metrics collected by the CloudWatch agent.
- Collect system-level metrics from on-premises servers. These can include servers in a hybrid environment as well as servers not managed by Amazon.
- Retrieve custom metrics from your applications or services using the `StatsD` and `collectd` protocols. `StatsD` is supported on both Linux servers and servers running Windows Server. `collectd` is supported only on Linux servers.
- Collect logs from Amazon EC2 instances and on-premises servers, running either Linux or Windows Server.

> **Note**
>
> The CloudWatch agent does not support collecting logs from FIFO pipes.

- Send the metrics to either CloudWatch or Amazon Managed Service for Prometheus, or to both. The CloudWatch agent configuration file contains a `metrics_destinations` parameter in the `metrics` section. You can specify `cloudwatch`, `amp`, or both in this parameter.
- Version 1.300031.0 and later can be used to enable CloudWatch Application Signals. For more information, see Application Signals.
- Version 1.300025.0 and later can collect traces from OpenTelemetry or X-Ray client SDKs, and send them to X-Ray.

  Using the CloudWatch agent allows you to collect traces without needing to run a separate trace collection daemon, helping to reduce the number of agents that you run and manage.

Metrics sent to CloudWatch can be viewed in CloudWatch just as any other CloudWatch metrics. The default CloudWatch namespace for metrics collected by the CloudWatch agent is `CWAgent`, although you can specify a different namespace when you configure the agent.

The logs collected by the unified CloudWatch agent are processed and stored in Amazon CloudWatch Logs, just like logs collected by the older CloudWatch Logs agent. For information about CloudWatch Logs pricing, see Amazon CloudWatch Pricing.

Metrics collected by the CloudWatch agent are billed as custom metrics. For more information about CloudWatch metrics pricing, see Amazon CloudWatch Pricing.

The CloudWatch agent is open-source under the MIT license, and is hosted on GitHub. If you would like to build, customize or contribute to the CloudWatch agent, see the GitHub repository for the latest instructions. If you think you've found a potential security issue, do not post it on GitHub or any public forum. Instead, please follow the instructions at Vulnerability Reporting or email Amazon security directly.

The steps in this section explain how to install the unified CloudWatch agent on Amazon EC2 instances and on-premises servers. For more information about the metrics that the CloudWatch agent can collect, see Metrics collected by the CloudWatch agent.

Supported operating systems

The CloudWatch agent is supported on x86-64 architecture on the following operating systems. It is also supported on all minor version updates for each of the major versions listed here.

- Amazon Linux 2023
- Amazon Linux 2
- Ubuntu Server versions 24.04, 23.10, 22.04, 20.04, 18.04, 16.04, and 14.04
- CentOS versions 9, 8, and 7
- Red Hat Enterprise Linux (RHEL) versions 9, 8, and 7
- Debian versions 12, 11 and 10
- SUSE Linux Enterprise Server (SLES) versions 15 and 12
- Oracle Linux versions 9, 8 and 7
- AlmaLinux versions 9 and 8
- Rocky Linux versions 9 and 8
- The following macOS computers: EC2 M1 Mac1 instances, and computers running macOS 14 (Sonoma), macOS 13 (Ventura), and macOS 12 (Monterey)
- Windows Server 2025, Windows Server 2022, Windows Server 2019, and Windows Server 2016

- Windows 11
- 64-bit Windows 10

The agent is supported on ARM64 architecture on the following operating systems. It is also supported on all minor version updates for each of the major versions listed here.

- Amazon Linux 2023
- Amazon Linux 2
- Ubuntu Server versions 23.10, 22.04, 20.04, 18.04, and 16.04
- CentOS versions 9 and 8
- Red Hat Enterprise Linux (RHEL) versions 9, 8, and 7
- Debian versions 12, 11 and 10
- SUSE Linux Enterprise Server 15
- The following macOS computers: macOS 14 (Sonoma), macOS 13 (Ventura), and macOS 12 (Monterey)

Installation process overview

You can download and install the CloudWatch agent manually using the command line, or you can integrate it with SSM. The general flow of installing the CloudWatch agent using either method is as follows:

1. Create IAM roles or users that enable the agent to collect metrics from the server and optionally to integrate with Amazon Systems Manager.
2. Download the agent package.
3. Modify the CloudWatch agent configuration file and specify the metrics that you want to collect.
4. Install and start the agent on your servers. As you install the agent on an EC2 instance, you attach the IAM role that you created in step 1. As you install the agent on an on-premises server, you specify a named profile that contains the credentials of the IAM user that you created in step 1.

Contents

- CloudWatch agent credentials preference
- Verifying the signature of the CloudWatch agent package
- Create the CloudWatch agent configuration file
- Install the CloudWatch agent with the Amazon CloudWatch Observability EKS add-on or the Helm chart
- Metrics collected by the CloudWatch agent
- Using the CloudWatch agent with related telemetry
- Common scenarios with the CloudWatch agent
- Troubleshooting the CloudWatch agent

Install the CloudWatch agent

You can download and install the CloudWatch agent using either the command line with an Amazon S3 download link, using SSM, or using an Amazon CloudFormation template.

Contents

- Install the CloudWatch agent using the command line
- Install the CloudWatch agent using Amazon Systems Manager
- Install the CloudWatch agent on on-premises servers

Install the CloudWatch agent using the command line

Use the following topics to download, configure, and install the CloudWatch agent package.

Topics

- Download and configure the CloudWatch agent using the command line
- Create IAM roles and users for use with CloudWatch agent
- Install and run the CloudWatch agent on your servers

Download and configure the CloudWatch agent using the command line

You can use the following steps to download the CloudWatch agent package, create IAM roles or users, and optionally modify the common configuration file.

Download the CloudWatch agent package

Note

To download the CloudWatch agent, your connection must use TLS 1.2 or later.

You can use an Amazon S3 download link to download the CloudWatch agent package. Choose the download link from this table, depending on your architecture and platform.

Architecture	Platform	Download link	Signature file link
amd64	Amazon Linux 2023 and Amazon Linux 2	https://s3.cn-north-1.amazonaws.com.cn/amazoncloudwatch-agent/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm	https://s3.cn-north-1.amazonaws.com.cn/amazoncloudwatch-agent/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm.sig
amd64	Centos	https://s3.cn-north-1.amazonaws.com.cn/amazoncloudwatch-agent/centos/amd64/latest/amazon-cloudwatch-agent.rpm	https://s3.cn-north-1.amazonaws.com.cn/amazoncloudwatch-agent/centos/amd64/latest/amazon-cloudwatch-agent.rpm.sig
amd64	Redhat	https://s3.cn-north-1.amazonaws.com.cn/amazoncloudwatch-agent/redhat/amd64/latest/amazon-cloudwatch-agent.rpm	https://s3.cn-north-1.amazonaws.com.cn/amazoncloudwatch-agent/redhat/amd64/latest/amazon-cloudwatch-agent.rpm.sig
amd64	SUSE	https://s3.cn-north-1.amazonaws.com.cn/amazoncloudwatch-agent/suse/amd64/latest/amazon-cloudwatch-agent.rpm	https://s3.cn-north-1.amazonaws.com.cn/amazoncloudwatch-agent/suse/amd64/latest/amazon-cloudwatch-agent.rpm.sig

Architecture	Platform	Download link	Signature file link
amd64	Debian	https://s3.cn-north-1.amazonaws.com.cn/amazoncloudwatch-agent/debian/amd64/latest/amazon-cloudwatch-agent.deb	https://s3.cn-north-1.amazonaws.com.cn/amazoncloudwatch-agent/debian/amd64/latest/amazon-cloudwatch-agent.deb.sig
amd64	Ubuntu	https://s3.cn-north-1.amazonaws.com.cn/amazoncloudwatch-agent/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb	https://s3.cn-north-1.amazonaws.com.cn/amazoncloudwatch-agent/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb.sig
amd64	Windows	https://s3.cn-north-1.amazonaws.com.cn/amazoncloudwatch-agent/windows/amd64/latest/amazon-cloudwatch-agent.msi	https://s3.cn-north-1.amazonaws.com.cn/amazoncloudwatch-agent/windows/amd64/latest/amazon-cloudwatch-agent.msi.sig
ARM64	Amazon Linux 2023 and Amazon Linux 2	https://s3.cn-north-1.amazonaws.com.cn/amazoncloudwatch-agent/amazon_linux/arm64/latest/amazon-cloudwatch-agent.rpm	https://s3.cn-north-1.amazonaws.com.cn/amazoncloudwatch-agent/amazon_linux/arm64/latest/amazon-cloudwatch-agent.rpm.sig
ARM64	Redhat	https://s3.cn-north-1.amazonaws.com.cn/amazoncloudwatch-agent/redhat/arm64/latest/amazon-cloudwatch-agent.rpm	https://s3.cn-north-1.amazonaws.com.cn/amazoncloudwatch-agent/redhat/arm64/latest/amazon-cloudwatch-agent.rpm.sig
ARM64	Ubuntu	https://s3.cn-north-1.amazonaws.com.cn/amazoncloudwatch-agent/ubuntu/arm64/latest/amazon-cloudwatch-agent.deb	https://s3.cn-north-1.amazonaws.com.cn/amazoncloudwatch-agent/ubuntu/arm64/latest/amazon-cloudwatch-agent.deb.sig

To use the command line to download and install the CloudWatch agent package

1. Download the CloudWatch agent.

   On a Linux server, enter the following. For *download-link*, use the appropriate download link from the previous table.

   ```
   wget download-link
   ```

   On a server running Windows Server, download the following file:

2. After you have downloaded the package, you can optionally verify the package signature. For more information, see Verifying the signature of the CloudWatch agent package.

3. Install the package. If you downloaded an RPM package on a Linux server, change to the directory containing the package and enter the following:

   ```
   sudo rpm -U ./amazon-cloudwatch-agent.rpm
   ```

   If you downloaded a DEB package on a Linux server, change to the directory containing the package and enter the following:

   ```
   sudo dpkg -i -E ./amazon-cloudwatch-agent.deb
   ```

   If you downloaded an MSI package on a server running Windows Server, change to the directory containing the package and enter the following:

   ```
   msiexec /i amazon-cloudwatch-agent.msi
   ```

   This command also works from within PowerShell. For more information about MSI command options, see Command-Line Options in the Microsoft Windows documentation.

   If you downloaded a PKG package on a macOS server, change to the directory containing the package and enter the following:

   ```
   sudo installer -pkg ./amazon-cloudwatch-agent.pkg -target /
   ```

Create and modify the agent configuration file

After you have downloaded the CloudWatch agent, you must create the configuration file before you start the agent on any servers. For more information, see Create the CloudWatch agent configuration file.

Create IAM roles and users for use with CloudWatch agent

Access to Amazon resources requires permissions. You create an IAM role, an IAM user, or both to grant permissions that the CloudWatch agent needs to write metrics to CloudWatch. If you're going to use the agent on Amazon EC2 instances, you must create an IAM role. If you're going to use the agent on on-premises servers, you must create an IAM user.

> **Note**
>
> We recently modified the following procedures by using new `CloudWatchAgentServerPolicy` and `CloudWatchAgentAdminPolicy` policies created by Amazon, instead of requiring customers to create these policies themselves. For writing files to and downloading files from the Parameter Store, the policies created by Amazon support only files with names that start with `AmazonCloudWatch-`. If you have a CloudWatch agent configuration file with a file name that doesn't start with `AmazonCloudWatch-`, these policies can't be used to write the file to Parameter Store or download it from Parameter Store.

If you're going to run the CloudWatch agent on Amazon EC2 instances, use the following steps to create the necessary IAM role. This role provides permissions for reading information from the instance and writing it to CloudWatch.

To create the IAM role necessary to run the CloudWatch agent on EC2 instances

1. Sign in to the Amazon Web Services Management Console and open the IAM console at https://console.amazonaws.cn/iam/.
2. In the navigation pane on the left, choose **Roles** and then **Create role**.
3. Make sure that **Amazon service** is selected under **Trusted entity type**.
4. For **Use case**, choose **EC2** under **Common use cases**,
5. Choose **Next**.

6. In the list of policies, select the check box next to **CloudWatchAgentServerPolicy**. If necessary, use the search box to find the policy.
7. (Optional) If the agent is sending traces to X-Ray you need to also give the role the **AWSXRayDaemonWriteAccess** policy. To do so, find that policy in the list and select the check box next to it.
8. Choose **Next**.
9. In **Role name**, enter a name for the role, such as *`CloudWatchAgentServerRole`*. Optionally give it a description. Then choose **Create role**.

   The role is now created.
10. (Optional) If the agent is going to send logs to CloudWatch Logs and you want the agent to be able to set retention policies for these log groups, you need to add the `logs:PutRetentionPolicy` permission to the role. For more information, see Allowing the CloudWatch agent to set log retention policy.

If you're going to run the CloudWatch agent on on-premises servers, use the following steps to create the necessary IAM user.

> **Warning**
>
> This scenario requires IAM users with programmatic access and long-term credentials, which presents a security risk. To help mitigate this risk, we recommend that you provide these users with only the permissions they require to perform the task and that you remove these users when they are no longer needed. Access keys can be updated if necessary. For more information, see Update access keys in the *IAM User Guide*.

To create the IAM user necessary for the CloudWatch agent to run on on-premises servers

1. Sign in to the Amazon Web Services Management Console and open the IAM console at https://console.amazonaws.cn/iam/.
2. In the navigation pane on the left, choose **Users** and then **Add users**.
3. Enter the user name for the new user.
4. Select **Access key - Programmatic access** and choose **Next: Permissions**.
5. Choose **Attach existing policies directly**.

6. In the list of policies, select the check box next to **CloudWatchAgentServerPolicy**. If necessary, use the search box to find the policy.
7. (Optional) If the agent is going to traces to X-Ray you need to also give the role the **AWSXRayDaemonWriteAccess** policy. To do so, find that policy in the list and select the check box next to it.
8. Choose **Next: Tags**.
9. Optionally create tags for the new IAM user, and then choose **Next:Review**.
10. Confirm that the correct policy is listed, and choose **Create user**.
11. Next to the name of the new user, choose **Show**. Copy the access key and secret key to a file so that you can use them when installing the agent. Choose **Close**.

Allowing the CloudWatch agent to set log retention policy

You can configure the CloudWatch agent to set the retention policy for log groups that it sends log events to. If you do this, you must grant the `logs:PutRetentionPolicy` to the IAM role or user that the agent uses. The agent uses an IAM role to run on Amazon EC2 instances, and uses an IAM user for on-premises servers.

To grant the CloudWatch agent's IAM role permission to set log retention policies

1. Sign in to the Amazon Web Services Management Console and open the IAM console at https://console.amazonaws.cn/iam/.
2. In the left navigation pane, choose **Roles**.
3. In the search box, Type the beginning of the name of the CloudWatch agent's IAM role. You chose this name when you created the role. It might be named `CloudWatchAgentServerRole`.

   When you see the role, choose the name of the role.
4. In the **Permissions** tab, choose **Add permissions**, **Create inline policy**.
5. Choose the **JSON** tab and copy the following policy into the box, replacing the default JSON in the box:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
```

```
      "Effect": "Allow",
      "Action": "logs:PutRetentionPolicy",
      "Resource": "*"
    }
  ]
}
```

6. Choose **Review policy**.
7. For **Name**, enter **`CloudWatchAgentPutLogsRetention`** or something similar, and choose **Create policy**.

To grant the CloudWatch agent's IAM user permission to set log retention policies

1. Sign in to the Amazon Web Services Management Console and open the IAM console at https://console.amazonaws.cn/iam/.
2. In the left navigation pane, choose **Users**.
3. In the search box, Type the beginning of the name of the CloudWatch agent's IAM user. You chose this name when you created the user.

   When you see the user, choose the name of the user.
4. In the **Permissions** tab, choose **Add inline policy**.
5. Choose the **JSON** tab and copy the following policy into the box, replacing the default JSON in the box:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "logs:PutRetentionPolicy",
      "Resource": "*"
    }
  ]
}
```

6. Choose **Review policy**.
7. For **Name**, enter **`CloudWatchAgentPutLogsRetention`** or something similar, and choose **Create policy**.

Install and run the CloudWatch agent on your servers

After you create your agent configuration file for an IAM role or IAM user, use the following steps to install and run the CloudWatch agent on your servers with that configuration. First, attach an IAM role or IAM user to the server that will run the agent. Then, on that server, download the agent package and start it using the agent configuration you created.

Download the CloudWatch agent package using an S3 download link

> **Note**
>
> To download the CloudWatch agent, your connection must use TLS 1.2 or later.

On each server where you will run the agent, download the agent package. Choose the download link from this table, depending on your architecture and platform.

Architecture	Platform	Download link	Signature file link
amd64	Amazon Linux 2023 and Amazon Linux 2	https://s3.cn-north-1.amazonaws.com.cn/amazoncloudwatch-agent/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm	https://s3.cn-north-1.amazonaws.com.cn/amazoncloudwatch-agent/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm.sig
amd64	Centos	https://s3.cn-north-1.amazonaws.com.cn/amazoncloudwatch-agent/centos/amd64/latest/amazon-cloudwatch-agent.rpm	https://s3.cn-north-1.amazonaws.com.cn/amazoncloudwatch-agent/centos/amd64/latest/amazon-cloudwatch-agent.rpm.sig
amd64	Redhat	https://s3.cn-north-1.amazonaws.com.cn/amazoncloudwatch-agent/redhat/amd64/latest/amazon-cloudwatch-agent.rpm	https://s3.cn-north-1.amazonaws.com.cn/amazoncloudwatch-agent/redhat/amd64/latest/amazon-cloudwatch-agent.rpm.sig

Architecture	Platform	Download link	Signature file link
amd64	SUSE	https://s3.cn-north-1.amazonaws.com.cn/amazoncloudwatch-agent/suse/amd64/latest/amazon-cloudwatch-agent.rpm	https://s3.cn-north-1.amazonaws.com.cn/amazoncloudwatch-agent/suse/amd64/latest/amazon-cloudwatch-agent.rpm.sig
amd64	Debian	https://s3.cn-north-1.amazonaws.com.cn/amazoncloudwatch-agent/debian/amd64/latest/amazon-cloudwatch-agent.deb	https://s3.cn-north-1.amazonaws.com.cn/amazoncloudwatch-agent/debian/amd64/latest/amazon-cloudwatch-agent.deb.sig
amd64	Ubuntu	https://s3.cn-north-1.amazonaws.com.cn/amazoncloudwatch-agent/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb	https://s3.cn-north-1.amazonaws.com.cn/amazoncloudwatch-agent/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb.sig
amd64	Windows	https://s3.cn-north-1.amazonaws.com.cn/amazoncloudwatch-agent/windows/amd64/latest/amazon-cloudwatch-agent.msi	https://s3.cn-north-1.amazonaws.com.cn/amazoncloudwatch-agent/windows/amd64/latest/amazon-cloudwatch-agent.msi.sig
ARM64	Amazon Linux 2023 and Amazon Linux 2	https://s3.cn-north-1.amazonaws.com.cn/amazoncloudwatch-agent/amazon_linux/arm64/latest/amazon-cloudwatch-agent.rpm	https://s3.cn-north-1.amazonaws.com.cn/amazoncloudwatch-agent/amazon_linux/arm64/latest/amazon-cloudwatch-agent.rpm.sig
ARM64	Redhat	https://s3.cn-north-1.amazonaws.com.cn/amazoncloudwatch-agent/redhat/arm64/latest/amazon-cloudwatch-agent.rpm	https://s3.cn-north-1.amazonaws.com.cn/amazoncloudwatch-agent/redhat/arm64/latest/amazon-cloudwatch-agent.rpm.sig

Architecture	Platform	Download link	Signature file link
ARM64	Ubuntu	https://s3.cn-north-1.amazonaws.com.cn/amazoncloudwatch-agent/ubuntu/arm64/latest/amazon-cloudwatch-agent.deb	https://s3.cn-north-1.amazonaws.com.cn/amazoncloudwatch-agent/ubuntu/arm64/latest/amazon-cloudwatch-agent.deb.sig

To use the command line to install the CloudWatch agent on an Amazon EC2 instance

1. Download the CloudWatch agent. For a Linux server, enter the following. For *download-link*, use the appropriate download link from the previous table.

```
wget download-link
```

   For a server running Windows Server, download the following file:

2. After you have downloaded the package, you can optionally verify the package signature. For more information, see Verifying the signature of the CloudWatch agent package.

3. Install the package. If you downloaded an RPM package on a Linux server, change to the directory containing the package and enter the following:

```
sudo rpm -U ./amazon-cloudwatch-agent.rpm
```

   If you downloaded a DEB package on a Linux server, change to the directory containing the package and enter the following:

```
sudo dpkg -i -E ./amazon-cloudwatch-agent.deb
```

   If you downloaded an MSI package on a server running Windows Server, change to the directory containing the package and enter the following:

```
msiexec /i amazon-cloudwatch-agent.msi
```

   This command also works from within PowerShell. For more information about MSI command options, see Command-Line Options in the Microsoft Windows documentation.

(Installing on an EC2 instance) Attaching an IAM role

To enable the CloudWatch agent to send data from the instance, you must attach an IAM role to the instance. The role to attach is **CloudWatchAgentServerRole**. You should have created this role previously. For more information see Create IAM roles and users for use with CloudWatch agent.

For more information on attaching an IAM role to an instance, see Attaching an IAM Role to an Instance in the *Amazon EC2 User Guide*.

(Installing on an on-premises server) Specify IAM credentials and Amazon Region

To enable the CloudWatch agent to send data from an on-premises server, you must specify the access key and secret key of the IAM user that you created earlier. For more information about creating this user, see Create IAM roles and users for use with CloudWatch agent.

You also must specify the Amazon Region to send the metrics to, using the `region` field in the `[AmazonCloudWatchAgent]` section of the Amazon config file, as in the following example.

```
[profile AmazonCloudWatchAgent]
region = us-west-1
```

The following is an example of using the `aws configure` command to create a named profile for the CloudWatch agent. This example assumes that you are using the default profile name of `AmazonCloudWatchAgent`.

To create the AmazonCloudWatchAgent profile for the CloudWatch agent

1. If you haven't already done so, install the Amazon Command Line Interface on the server. For more information, see Installing the Amazon CLI.
2. On Linux servers, enter the following command and follow the prompts:

   ```
   sudo aws configure --profile AmazonCloudWatchAgent
   ```

   On Windows Server, open PowerShell as an administrator, enter the following command, and follow the prompts.

   ```
   aws configure --profile AmazonCloudWatchAgent
   ```

Verify internet access

Your Amazon EC2 instances must have outbound internet access to send data to CloudWatch or CloudWatch Logs. For more information about how to configure internet access, see Internet Gateways in the *Amazon VPC User Guide*.

The endpoints and ports to configure on your proxy are as follows:

- If you're using the agent to collect metrics, you must add the CloudWatch endpoints for the appropriate Regions to the allow list. These endpoints are listed in Amazon CloudWatch endpoints and quotas.
- If you're using the agent to collect logs, you must add the CloudWatch Logs endpoints for the appropriate Regions to the allow list. These endpoints are listed in Amazon CloudWatch Logs endpoints and quotas.
- If you're using Systems Manager to install the agent or Parameter Store to store your configuration file, you must add the Systems Manager endpoints for the appropriate Regions to the allow list. These endpoints are listed in Amazon Systems Manager endpoints and quotas.

(Optional) Modify the common configuration for proxy or Region information

The CloudWatch agent includes a configuration file called `common-config.toml`. You can optionally use this file to specify proxy and Region information.

On a server running Linux, this file is in the `/opt/aws/amazon-cloudwatch-agent/etc` directory. On a server running Windows Server, this file is in the `C:\ProgramData\Amazon\AmazonCloudWatchAgent` directory.

> **Note**
>
> We recommend that you use the `common-config.toml` file to provide shared configuration and credentials when you run the CloudWatch agent in an on-premise mode, and it can also be useful when you are running on Amazon EC2 and you want to reuse existing shared credential profiles and files. Enabling it via the `common-config.toml` has the added advantage that if your shared credentials file gets rotated with renewed credentials after they expire, the new credentials are automatically picked up by the agent without requiring a restart.

The default `common-config.toml` is as follows.

```
# This common-config is used to configure items used for both ssm and cloudwatch access


## Configuration for shared credential.
## Default credential strategy will be used if it is absent here:
##            Instance role is used for EC2 case by default.
##            AmazonCloudWatchAgent profile is used for the on-premises case by
 default.
# [credentials]
#    shared_credential_profile = "{profile_name}"
#    shared_credential_file= "{file_name}"

## Configuration for proxy.
## System-wide environment-variable will be read if it is absent here.
## i.e. HTTP_PROXY/http_proxy; HTTPS_PROXY/https_proxy; NO_PROXY/no_proxy
## Note: system-wide environment-variable is not accessible when using ssm run-command.
## Absent in both here and environment-variable means no proxy will be used.
# [proxy]
#    http_proxy = "{http_url}"
#    https_proxy = "{https_url}"
#    no_proxy = "{domain}"
```

All lines are commented out initially. To set the credential profile or proxy settings, remove the # from that line and specify a value. You can edit this file manually or by using the `RunShellScript` Run Command in Systems Manager:

- `shared_credential_profile` – For on-premises servers, this line specifies the IAM user credential profile to use to send data to CloudWatch. If you keep this line commented out, `AmazonCloudWatchAgent` is used. For more information about creating this profile, see [(Installing on an on-premises server) Specify IAM credentials and Amazon Region](#).

  On an EC2 instance, you can use this line to have the CloudWatch agent send data from this instance to CloudWatch in a different Amazon Region. To do so, specify a named profile that includes a `region` field specifying the name of the Region to send to.

  If you specify a `shared_credential_profile`, you must also remove the # from the beginning of the `[credentials]` line.

- `shared_credential_file` – To have the agent look for credentials in a file located in a path other than the default path, specify that complete path and file name here. The default path is `/root/.aws` on Linux and is `C:\\Users\\Administrator\\.aws` on Windows Server.

  The first example below shows the syntax of a valid `shared_credential_file` line for Linux servers, and the second example is valid for Windows Server. On Windows Server, you must escape the \ characters.

  ```
  shared_credential_file= "/usr/username/credentials"
  ```

  ```
  shared_credential_file= "C:\\Documents and Settings\\username\\.aws\\credentials"
  ```

  If you specify a `shared_credential_file`, you must also remove the # from the beginning of the `[credentials]` line.

- Proxy settings – If your servers use HTTP or HTTPS proxies to contact Amazon services, specify those proxies in the `http_proxy` and `https_proxy` fields. If there are URLs that should be excluded from proxying, specify them in the `no_proxy` field, separated by commas.

Start the CloudWatch agent using the command line

Follow these steps to use the command line to start the CloudWatch agent on a server.

For information about setting up the agent on a system that has security-enhanced Linux (SELinux) enabled, see Set up the CloudWatch agent with security-enhanced Linux (SELinux).

To use the command line to start the CloudWatch agent on a server

1. Copy the agent configuration file that you want to use to the server where you're going to run the agent. Note the pathname where you copy it to.
2. In this command, `-a fetch-config` causes the agent to load the latest version of the CloudWatch agent configuration file, and `-s` starts the agent.

   Enter one of the following commands. Replace *`configuration-file-path`* with the path to the agent configuration file. This file is called `config.json` if you created it with the wizard, and might be called `amazon-cloudwatch-agent.json` if you created it manually.

   On an EC2 instance running Linux, enter the following command.

```
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-
config -m ec2 -s -c file:configuration-file-path
```

On an on-premises server running Linux, enter the following:

```
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-
config -m onPremise -s -c file:configuration-file-path
```

On an EC2 instance running Windows Server, enter the following from the PowerShell console:

```
& "C:\Program Files\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1" -
a fetch-config -m ec2 -s -c file:configuration-file-path
```

On an on-premises server running Windows Server, enter the following from the PowerShell console:

```
& "C:\Program Files\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1" -
a fetch-config -m onPremise -s -c file:configuration-file-path
```

Install the CloudWatch agent using Amazon Systems Manager

Using Amazon Systems Manager makes it easier to install the CloudWatch agent on a fleet of Amazon EC2 instances. You can download the agent into one server and create your CloudWatch agent configuration file for all servers in the fleet. Then you can use Systems Manager to install the agent on the other servers, using the configuration file that you created. Use the following topics to install and run the CloudWatch agent using Amazon Systems Manager.

Topics

Create IAM roles and users for use with the CloudWatch agent

Access to Amazon resources requires permissions. You can create IAM roles and users that include the permissions that you need for the CloudWatch agent to write metrics to CloudWatch and for the CloudWatch agent to communicate with Amazon EC2 and Amazon Systems Manager. You use IAM roles on Amazon EC2 instances, and you use IAM users with on-premises servers.

One role or user enables CloudWatch agent to be installed on a server and send metrics to CloudWatch. The other role or user is needed to store your CloudWatch agent configuration in Systems Manager Parameter Store. Parameter Store enables multiple servers to use one CloudWatch agent configuration.

The ability to write to Parameter Store is a broad and powerful permission. You should use it only when you need it, and it shouldn't be attached to multiple instances in your deployment. If you store your CloudWatch agent configuration in Parameter Store, we recommend the following:

- Set up one instance where you perform this configuration.
- Use the IAM role with permissions to write to Parameter Store only on this instance.
- Use the IAM role with permissions to write to Parameter Store only while you are working with and saving the CloudWatch agent configuration file.

> **Note**
>
> We recently modified the following procedures by using new `CloudWatchAgentServerPolicy` and `CloudWatchAgentAdminPolicy` policies created by Amazon, instead of requiring customers to create these policies themselves. To use these policies to write the agent configuration file to Parameter Store and then download it from Parameter Store, your agent configuration file must have a name that starts with `AmazonCloudWatch-`. If you have a CloudWatch agent configuration file with a file name that doesn't start with `AmazonCloudWatch-`, these policies can't be used to write the file to Parameter Store or to download the file from Parameter Store.

Create IAM roles to use with the CloudWatch agent on Amazon EC2 instances

The first procedure creates the IAM role that you must attach to each Amazon EC2 instance that runs the CloudWatch agent. This role provides permissions for reading information from the instance and writing it to CloudWatch.

The second procedure creates the IAM role that you must attach to the Amazon EC2 instance being used to create the CloudWatch agent configuration file. This step is necessary if you're going to store this file in Systems Manager Parameter Store so that other servers can use it. This role provides permissions for writing to Parameter Store, in addition to the permissions for reading information from the instance and writing it to CloudWatch. This role includes permissions sufficient to run the CloudWatch agent as well as to write to Parameter Store.

> **Note**
>
> Parameter Store supports parameters in Standard and Advanced tiers. These parameter tiers are not related to the Basic, Standard, and Advanced levels of details available with the CloudWatch Agent predefined metric sets.

To create the IAM role necessary for each server to run the CloudWatch agent

1. Sign in to the Amazon Web Services Management Console and open the IAM console at https://console.amazonaws.cn/iam/.
2. In the navigation pane, choose **Roles** and then choose **Create role**.
3. Under **Select type of trusted entity**, choose **Amazon service**.
4. Immediately under **Common use cases**, choose **EC2**,and then choose **Next: Permissions**.
5. In the list of policies, select the check box next to **CloudWatchAgentServerPolicy**. If necessary, use the search box to find the policy.
6. To use Systems Manager to install or configure the CloudWatch agent, select the box next to **AmazonSSMManagedInstanceCore**. This Amazon managed policy enables an instance to use Systems Manager service core functionality. If necessary, use the search box to find the policy. This policy isn't necessary if you start and configure the agent only through the command line.
7. Choose **Next: Tags**.
8. (Optional) Add one or more tag-key value pairs to organize, track, or control access for this role, and then choose **Next: Review**.
9. For **Role name**, enter a name for your new role, such as `CloudWatchAgentServerRole` or another name that you prefer.
10. (Optional) For **Role description**, enter a description.
11. Confirm that **CloudWatchAgentServerPolicy** and optionally **AmazonSSMManagedInstanceCore** appear next to **Policies**.
12. Choose **Create role**.

The role is now created.

The following procedure creates the IAM role that can also write to Parameter Store. You can use this role to store the agent configuration file in Parameter Store so that other servers can retrieve it.

The permissions for writing to Parameter Store provide broad access. This role shouldn't be attached to all your servers, and only administrators should use it. After you create the agent configuration file and copy it to Parameter Store, you should detach this role from the instance and use `CloudWatchAgentServerRole` instead.

To create the IAM role for an administrator to write to Parameter Store

1. Sign in to the Amazon Web Services Management Console and open the IAM console at https://console.amazonaws.cn/iam/.
2. In the navigation pane, choose **Roles** and then choose **Create role**.
3. Under **Select type of trusted entity**, choose **Amazon service**.
4. Immediately under **Choose the service that will use this role**, choose **EC2**, and then choose **Next: Permissions**.
5. In the list of policies, select the check box next to **CloudWatchAgentAdminPolicy**. If necessary, use the search box to find the policy.
6. To use Systems Manager to install or configure the CloudWatch agent, select the box next to **AmazonSSMManagedInstanceCore**. This Amazon managed policy enables an instance to use Systems Manager service core functionality. If necessary, use the search box to find the policy. This policy isn't necessary if you start and configure the agent only through the command line.
7. Choose **Next: Tags**.
8. (Optional) Add one or more tag-key value pairs to organize, track, or control access for this role, and then choose **Next: Review**.
9. For **Role name**, enter a name for your new role, such as **`CloudWatchAgentAdminRole`** or another name that you prefer.
10. (Optional) For **Role description**, enter a description.
11. Confirm that **CloudWatchAgentAdminPolicy** and optionally **AmazonSSMManagedInstanceCore** appear next to **Policies**.
12. Choose **Create role**.

The role is now created.

Create IAM users to use with the CloudWatch agent on on-premises servers

The first procedure creates the IAM user that you need to run the CloudWatch agent. This user provides permissions to send data to CloudWatch.

The second procedure creates the IAM user that you can use when creating the CloudWatch agent configuration file. Use this procedure to store this file in Systems Manager Parameter Store so that other servers can use it. This user provides permissions to write to Parameter Store, in addition to the permissions to write data to CloudWatch.

> **Note**
>
> Parameter Store supports parameters in Standard and Advanced tiers. These parameter tiers are not related to the Basic, Standard, and Advanced levels of details available with the CloudWatch Agent predefined metric sets.

To create the IAM user necessary for the CloudWatch agent to write data to CloudWatch

1. Sign in to the Amazon Web Services Management Console and open the IAM console at https://console.amazonaws.cn/iam/.
2. In the navigation pane, choose **Users**, and then choose **Add user**.
3. Enter the user name for the new user.
4. For **Access type**, select **Programmatic access**, and then choose **Next: Permissions**.
5. For **Set permissions**, choose **Attach existing policies directly**.
6. In the list of policies, select the check box next to **CloudWatchAgentServerPolicy**. If necessary, use the search box to find the policy.
7. To use Systems Manager to install or configure the CloudWatch agent, select the box next to **AmazonSSMManagedInstanceCore**. This Amazon managed policy enables an instance to use Systems Manager service core functionality. (If necessary, use the search box to find the policy. This policy isn't necessary if you start and configure the agent only through the command line.)
8. Choose **Next: Tags**.

9. (Optional) Add one or more tag-key value pairs to organize, track, or control access for this role, and then choose **Next: Review**.
10. Confirm that the correct policies are listed, and then choose **Create user**.
11. In the row for the new user, choose **Show**. Copy the access key and secret key to a file so that you can use them when installing the agent. Choose **Close**.

The following procedure creates the IAM user that can also write to Parameter Store. If you're going to store the agent configuration file in Parameter Store so that other servers can use it, you need to use this IAM user. This IAM user provides permissions for writing to Parameter Store. This user also provides the permissions for reading information from the instance and writing it to CloudWatch. The permissions for writing to Systems Manager Parameter Store provide broad access. This IAM user shouldn't be attached to all your servers, and only administrators should use it. You should use this IAM user only when you are storing the agent configuration file in Parameter Store.

To create the IAM user necessary to store the configuration file in Parameter Store and send information to CloudWatch

1. Sign in to the Amazon Web Services Management Console and open the IAM console at https://console.amazonaws.cn/iam/.
2. In the navigation pane, choose **Users**, and then choose **Add user**.
3. Enter the user name for the new user.
4. For **Access type**, select **Programmatic access**, and then choose **Next: Permissions**.
5. For **Set permissions**, choose **Attach existing policies directly**.
6. In the list of policies, select the check box next to **CloudWatchAgentAdminPolicy**. If necessary, use the search box to find the policy.
7. To use Systems Manager to install or configure the CloudWatch agent, select the check box next to **AmazonSSMManagedInstanceCore**. This Amazon managed policy enables an instance to use Systems Manager service core functionality. (If necessary, use the search box to find the policy. This policy isn't necessary if you start and configure the agent only through the command line.)
8. Choose **Next: Tags**.
9. (Optional) Add one or more tag-key value pairs to organize, track, or control access for this role, and then choose **Next: Review**.
10. Confirm that the correct policies are listed, and then choose **Create user**.

11. In the row for the new user, choose **Show**. Copy the access key and secret key to a file so that you can use them when installing the agent. Choose **Close**.

Download, configure, and run the CloudWatch agent using SSM

This section explains how to use Systems Manager to download the agent and then how to create your agent configuration file. Before you can use Systems Manager to download the agent, you must make sure that the instance is configured correctly for Systems Manager.

Install or update the SSM Agent

On an Amazon EC2 instance, the CloudWatch agent requires that the instance is running version 2.2.93.0 or later of the SSM Agent. Before you install the CloudWatch agent, update or install SSM Agent on the instance if you haven't already done so.

For information about installing or updating SSM Agent on an instance running Linux, see Installing and Configuring SSM Agent on Linux Instances in the *Amazon Systems Manager User Guide*.

For information about installing or updating the SSM Agent, see Working with SSM Agent in the *Amazon Systems Manager User Guide*.

Verify Systems Manager prerequisites

Before you use Systems Manager Run Command to install and configure the CloudWatch agent, verify that your instances meet the minimum Systems Manager requirements. For more information, see Systems Manager Prerequisites in the *Amazon Systems Manager User Guide*.

Verify internet access

Your Amazon EC2 instances must be able to connect to CloudWatch endpoints. This can be by Internet Gateway, NAT gateway, or CloudWatch Interface VPC endpoints. For more information about how to configure internet access, see Internet Gateways in the *Amazon VPC User Guide*.

The endpoints and ports to configure on your proxy are as follows:

- If you're using the agent to collect metrics, you must allow list the CloudWatch endpoints for the appropriate Regions. These endpoints are listed in Amazon CloudWatch in the *Amazon Web Services General Reference*.

- If you're using the agent to collect logs, you must allow list the CloudWatch Logs endpoints for the appropriate Regions. These endpoints are listed in Amazon CloudWatch Logs in the *Amazon Web Services General Reference*.
- If you're using Systems Manager to install the agent or Parameter Store to store your configuration file, you must allow list the Systems Manager endpoints for the appropriate Regions. These endpoints are listed in Amazon Systems Manager in the *Amazon Web Services General Reference*.

Download the CloudWatch agent package to your first instance

Use the following steps to download the CloudWatch agent package using Systems Manager.

To download the CloudWatch agent using Systems Manager

1. Open the Systems Manager console at https://console.amazonaws.cn/systems-manager/.
2. In the navigation pane, choose **Run Command**.

   -or-

   If the Amazon Systems Manager home page opens, scroll down and choose **Explore Run Command**.
3. Choose **Run command**.
4. In the **Command document** list, choose **AWS-ConfigureAWSPackage**.
5. In the **Targets** area, choose the instance to install the CloudWatch agent on. If you don't see a specific instance, it might not be configured as a managed instance for use with Systems Manager. For more information, see Setting Up Amazon Systems Manager for Hybrid Environments in the *Amazon Systems Manager User Guide*.
6. In the **Action** list, choose **Install**.
7. In the **Name** field, enter *`AmazonCloudWatchAgent`*.
8. Keep **Version** set to **latest** to install the latest version of the agent.
9. Choose **Run**.
10. Optionally, in the **Targets and outputs** areas, select the button next to an instance name and choose **View output**. Systems Manager should show that the agent was successfully installed.

Create and modify the agent configuration file

After you have downloaded the CloudWatch agent, you must create the configuration file before you start the agent on any servers.

If you're going to save your agent configuration file in the Systems Manager Parameter Store, you must use an EC2 instance to save to the Parameter Store. Additionally, you must first attach to that instance the `CloudWatchAgentAdminRole` IAM role. For more information about attaching roles, see Attaching an IAM Role to an Instance in the *Amazon EC2 User Guide*.

For more information about creating the CloudWatch agent configuration file, see Create the CloudWatch agent configuration file.

Install and start the CloudWatch agent on additional EC2 instances using your agent configuration

After you have a CloudWatch agent configuration saved in Parameter Store, you can use it when you install the agent on other servers.

For each of these servers, follow the steps listed previously in this section to verify the Systems Manager prerequisites, the version of the SSM Agent, and internet access. Then use the following instructions to install the CloudWatch agent on the additional instances, using the CloudWatch agent configuration file that you have created.

Step 1: Download and install the CloudWatch agent

To be able to send the CloudWatch data to a different Region, make sure that the IAM role that you attached to this instance has permissions to write the CloudWatch data in that Region.

Following is an example of using the `aws configure` command to create a named profile for the CloudWatch agent. This example assumes that you are using the default profile name of `AmazonCloudWatchAgent`.

To create the AmazonCloudWatchAgent profile for the CloudWatch agent

- On Linux servers, type the following command and follow the prompts:

  ```
  sudo aws configure --profile AmazonCloudWatchAgent
  ```

  On Windows Server, open PowerShell as an administrator, type the following command and follow the prompts.

```
aws configure --profile AmazonCloudWatchAgent
```

Install the CloudWatch agent on additional EC2 instances using your agent configuration

After you have a CloudWatch agent configuration saved in Parameter Store, you can use it when you install the agent on other servers.

For each of these servers, follow the steps listed previously in this section to verify the Systems Manager prerequisites, the version of the SSM Agent, and internet access. Then use the following instructions to install the CloudWatch agent on the additional instances, using the CloudWatch agent configuration file that you have created.

Step 1: Download and install the CloudWatch agent

> **Note**
>
> When you install or update the CloudWatch agent, only the **Uninstall and reinstall** option is supported. You can't use the **In-place update** option.

You can download the CloudWatch agent package using either Systems Manager Run Command or an Amazon S3 download link. For information about using an Amazon S3 download link, see Download the CloudWatch agent package.

Systems Manager Run Command enables you to manage the configuration of your instances. You specify a Systems Manager document, specify parameters, and execute the command on one or more instances. SSM Agent on the instance processes the command and configures the instance as specified.

To download the CloudWatch agent using Run Command

1. Open the Systems Manager console at https://console.amazonaws.cn/systems-manager/.
2. In the navigation pane, choose **Run Command**.

   -or-

   If the Amazon Systems Manager home page opens, scroll down and choose **Explore Run Command**.

3. Choose **Run command**.
4. In the **Command document** list, choose **AWS-ConfigureAWSPackage**.
5. In the **Targets** area, choose the instance on which to install the CloudWatch agent. If you do not see a specific instance, it might not be configured for Run Command. For more information, see Setting Up Amazon Systems Manager for Hybrid Environments in the *Amazon Systems Manager User Guide*.
6. In the **Action** list, choose **Install**.
7. In the **Name** box, enter *AmazonCloudWatchAgent*.
8. Keep **Version** set to **latest** to install the latest version of the agent.
9. Choose **Run**.
10. Optionally, in the **Targets and outputs** areas, select the button next to an instance name and choose **View output**. Systems Manager should show that the agent was successfully installed.

Step 2: Start the CloudWatch agent using your agent configuration file

Follow these steps to start the agent using Systems Manager Run Command.

For information about setting up the agent on a system that has security-enhanced Linux (SELinux) enabled, see Set up the CloudWatch agent with security-enhanced Linux (SELinux).

To start the CloudWatch agent using Run Command

1. Open the Systems Manager console at https://console.amazonaws.cn/systems-manager/.
2. In the navigation pane, choose **Run Command**.

   -or-

   If the Amazon Systems Manager home page opens, scroll down and choose **Explore Run Command**.
3. Choose **Run command**.
4. In the **Command document** list, choose **AmazonCloudWatch-ManageAgent**.
5. In the **Targets** area, choose the instance where you installed the CloudWatch agent.
6. In the **Action** list, choose **configure**.
7. In the **Optional Configuration Source** list, choose **ssm**.

8. In the **Optional Configuration Location** box, enter the name of the Systems Manager parameter name of the agent configuration file that you created and saved to Systems Manager Parameter Store, as explained in [Create the CloudWatch agent configuration file](#).
9. In the **Optional Restart** list, choose **yes** to start the agent after you have finished these steps.
10. Choose **Run**.
11. Optionally, in the **Targets and outputs** areas, select the button next to an instance name and choose **View output**. Systems Manager should show that the agent was successfully started.

(Optional) Modify the common configuration and named profile for CloudWatch agent

The CloudWatch agent includes a configuration file called `common-config.toml`. You can use this file to optionally specify proxy and Region information.

On a server running Linux, this file is in the `/opt/aws/amazon-cloudwatch-agent/etc` directory. On a server running Windows Server, this file is in the `C:\ProgramData\Amazon\AmazonCloudWatchAgent` directory.

The default `common-config.toml` is as follows:

```
# This common-config is used to configure items used for both ssm and cloudwatch access


## Configuration for shared credential.
## Default credential strategy will be used if it is absent here:
##            Instance role is used for EC2 case by default.
##            AmazonCloudWatchAgent profile is used for onPremise case by default.
# [credentials]
#    shared_credential_profile = "{profile_name}"
#    shared_credential_file= "{file_name}"

## Configuration for proxy.
## System-wide environment-variable will be read if it is absent here.
## i.e. HTTP_PROXY/http_proxy; HTTPS_PROXY/https_proxy; NO_PROXY/no_proxy
## Note: system-wide environment-variable is not accessible when using ssm run-command.
## Absent in both here and environment-variable means no proxy will be used.
# [proxy]
#    http_proxy = "{http_url}"
#    https_proxy = "{https_url}"
#    no_proxy = "{domain}"
```

All lines are commented out initially. To set the credential profile or proxy settings, remove the # from that line and specify a value. You can edit this file manually, or by using the `RunShellScript` Run Command in Systems Manager:

- `shared_credential_profile` – For on-premises servers, this line specifies the IAM user credential profile to use to send data to CloudWatch. If you keep this line commented out, `AmazonCloudWatchAgent` is used.

  On an EC2 instance, you can use this line to have the CloudWatch agent send data from this instance to CloudWatch in a different Amazon Region. To do so, specify a named profile that includes a `region` field specifying the name of the Region to send to.

  If you specify a `shared_credential_profile`, you must also remove the # from the beginning of the `[credentials]` line.
- `shared_credential_file` – To have the agent look for credentials in a file located in a path other than the default path, specify that complete path and file name here. The default path is `/root/.aws` on Linux and is `C:\\Users\\Administrator\\.aws` on Windows Server.

  The first example below shows the syntax of a valid `shared_credential_file` line for Linux servers, and the second example is valid for Windows Server. On Windows Server, you must escape the \ characters.

  ```
  shared_credential_file= "/usr/username/credentials"
  ```

  ```
  shared_credential_file= "C:\\Documents and Settings\\username\\.aws\\credentials"
  ```

  If you specify a `shared_credential_file`, you must also remove the # from the beginning of the `[credentials]` line.
- Proxy settings – If your servers use HTTP or HTTPS proxies to contact Amazon services, specify those proxies in the `http_proxy` and `https_proxy` fields. If there are URLs that should be excluded from proxying, specify them in the `no_proxy` field, separated by commas.

Install the CloudWatch agent on on-premises servers

If you downloaded the CloudWatch agent on a computer and created your agent configuration file, you can use that configuration file to install the agent in other on-premises servers.

Download the CloudWatch agent on an on-premises server

You can download the CloudWatch agent package using either Systems Manager Run Command or an Amazon S3 download link. For information about using an Amazon S3 download link, see Download the CloudWatch agent package.

Download using Systems Manager

To use Systems Manager Run Command, you must register your on-premises server with Amazon EC2 Systems Manager. For more information, see Setting Up Systems Manager in Hybrid Environments in the *Amazon Systems Manager User Guide*.

If you have already registered your server, update SSM Agent to the latest version.

For information about updating SSM Agent on a server running Linux, see Install SSM Agent for a Hybrid Environment (Linux) in the *Amazon Systems Manager User Guide*.

For information about updating the SSM Agent on a server running Windows Server, see Install SSM Agent for a Hybrid Environment (Windows) in the *Amazon Systems Manager User Guide*.

To use the SSM Agent to download the CloudWatch agent package on an on-premises server

1. Open the Systems Manager console at https://console.amazonaws.cn/systems-manager/.
2. In the navigation pane, choose **Run Command**.

   -or-

   If the Amazon Systems Manager home page opens, scroll down and choose **Explore Run Command**.
3. Choose **Run command**.
4. In the **Command document** list, select the button next to **AWS-ConfigureAWSPackage**.
5. In the **Targets** area, select the server to install the CloudWatch agent on. If you don't see a specific server, it might not be configured for Run Command. For more information, see Setting Up Amazon Systems Manager for Hybrid Environments in the *Amazon Systems Manager User Guide*.
6. In the **Action** list, choose **Install**.
7. In the **Name** box, enter *`AmazonCloudWatchAgent`*.
8. Keep **Version** blank to install the latest version of the agent.
9. Choose **Run**.

The agent package is downloaded, and the next steps are to configure and start it.

(Installing on an on-premises server) Specify IAM credentials and Amazon Region

To enable the CloudWatch agent to send data from an on-premises server, you must specify the access key and secret key of the IAM user that you created earlier. For more information about creating this user, see Create IAM roles and users for use with the CloudWatch agent.

You also must specify the Amazon Region to send the metrics to, using the `region` field.

Following is an example of this file.

```
[AmazonCloudWatchAgent]
aws_access_key_id=my_access_key
aws_secret_access_key=my_secret_key
region = us-west-1
```

For *`my_access_key`* and *`my_secret_key`*, use the keys from the IAM user that doesn't have the permissions to write to Systems Manager Parameter Store. For more information about the IAM users needed for CloudWatch agent, see Create IAM users to use with the CloudWatch agent on on-premises servers.

If you name this profile `AmazonCloudWatchAgent`, you don't need to do anything more. Optionally, you can give it a different name and specify that name as the value for `shared_credential_profile` in the `common-config.toml` file, which is explained in the following section.

Following is an example of using the **aws configure** command to create a named profile for the CloudWatch agent. This example assumes that you're using the default profile name of `AmazonCloudWatchAgent`.

To create the AmazonCloudWatchAgent profile for the CloudWatch agent

1. If you haven't already done so, install the Amazon Command Line Interface on the server. For more information, see Installing the Amazon CLI.
2. On Linux servers, enter the following command and follow the prompts:

   ```
   sudo aws configure --profile AmazonCloudWatchAgent
   ```

On Windows Server, open PowerShell as an administrator, enter the following command, and follow the prompts.

```
aws configure --profile AmazonCloudWatchAgent
```

(Optional) Modifying the common configuration and named profile for CloudWatch agent

The CloudWatch agent includes a configuration file called `common-config.toml`. You can optionally use this file to specify proxy and Region information.

On a server running Linux, this file is in the `/opt/aws/amazon-cloudwatch-agent/etc` directory. On a server running Windows Server, this file is in the `C:\ProgramData\Amazon\AmazonCloudWatchAgent` directory.

The default `common-config.toml` is as follows:

```
# This common-config is used to configure items used for both ssm and cloudwatch access


## Configuration for shared credential.
## Default credential strategy will be used if it is absent here:
##            Instance role is used for EC2 case by default.
##            AmazonCloudWatchAgent profile is used for onPremise case by default.
# [credentials]
#    shared_credential_profile = "{profile_name}"
#    shared_credential_file= "{file_name}"

## Configuration for proxy.
## System-wide environment-variable will be read if it is absent here.
## i.e. HTTP_PROXY/http_proxy; HTTPS_PROXY/https_proxy; NO_PROXY/no_proxy
## Note: system-wide environment-variable is not accessible when using ssm run-command.
## Absent in both here and environment-variable means no proxy will be used.
# [proxy]
#    http_proxy = "{http_url}"
#    https_proxy = "{https_url}"
#    no_proxy = "{domain}"
```

All lines are commented out initially. To set the credential profile or proxy settings, remove the # from that line and specify a value. You can edit this file manually, or by using the `RunShellScript` Run Command in Systems Manager:

- `shared_credential_profile` – For on-premises servers, this line specifies the IAM user credential profile to use to send data to CloudWatch. If you keep this line commented out, `AmazonCloudWatchAgent` is used. For more information about creating this profile, see [(Installing on an on-premises server) Specify IAM credentials and Amazon Region](#).

  On an EC2 instance, you can use this line to have the CloudWatch agent send data from this instance to CloudWatch in a different Amazon Region. To do so, specify a named profile that includes a `region` field specifying the name of the Region to send to.

  If you specify a `shared_credential_profile`, you must also remove the # from the beginning of the `[credentials]` line.
- `shared_credential_file` – To have the agent look for credentials in a file located in a path other than the default path, specify that complete path and file name here. The default path is `/root/.aws` on Linux and is `C:\\Users\\Administrator\\.aws` on Windows Server.

  The first example below shows the syntax of a valid `shared_credential_file` line for Linux servers, and the second example is valid for Windows Server. On Windows Server, you must escape the \ characters.

  ```
  shared_credential_file= "/usr/username/credentials"
  ```

  ```
  shared_credential_file= "C:\\Documents and Settings\\username\\.aws\\credentials"
  ```

  If you specify a `shared_credential_file`, you must also remove the # from the beginning of the `[credentials]` line.
- Proxy settings – If your servers use HTTP or HTTPS proxies to contact Amazon services, specify those proxies in the `http_proxy` and `https_proxy` fields. If there are URLs that should be excluded from proxying, specify them in the `no_proxy` field, separated by commas.

Starting the CloudWatch agent

You can start the CloudWatch agent using either Systems Manager Run Command or the command line.

For information about setting up the agent on a system that has security-enhanced Linux (SELinux) enabled, see Set up the CloudWatch agent with security-enhanced Linux (SELinux).

To use SSM Agent to start the CloudWatch agent on an on-premises server

1. Open the Systems Manager console at https://console.amazonaws.cn/systems-manager/.
2. In the navigation pane, choose **Run Command**.

   -or-

   If the Amazon Systems Manager home page opens, scroll down and choose **Explore Run Command**.
3. Choose **Run command**.
4. In the **Command document** list, select the button next to **AmazonCloudWatch-ManageAgent**.
5. In the **Targets** area, select the instance where you installed the agent.
6. In the **Action** list, choose **configure**.
7. In the **Mode** list, choose **onPremise**.
8. In the **Optional Configuration Location** box, enter the name of the agent configuration file that you created with the wizard and stored in the Parameter Store.
9. Choose **Run**.

   The agent starts with the configuration you specified in the configuration file.

To use the command line to start the CloudWatch agent on an on-premises server

- In this command, `-a fetch-config` causes the agent to load the latest version of the CloudWatch agent configuration file, and `-s` starts the agent.

  Linux: If you saved the configuration file in the Systems Manager Parameter Store, enter the following:

  ```
  sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-config -m onPremise -s -c ssm:configuration-parameter-store-name
  ```

  Linux: If you saved the configuration file on the local computer, enter the following command. Replace *`configuration-file-path`* with the path to the agent configuration file. This file is called `config.json` if you created it with the wizard, and might be called `amazon-cloudwatch-agent.json` if you created it manually.

```
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-config -m onPremise -s -c file:configuration-file-path
```

Windows Server: If you saved the agent configuration file in Systems Manager Parameter Store, enter the following from the PowerShell console:

```
& "C:\Program Files\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1" -a fetch-config -m onPremise -s -c ssm:configuration-parameter-store-name
```

Windows Server: If you saved the agent configuration file on the local computer, enter the following from the PowerShell console. Replace *`configuration-file-path`* with the path to the agent configuration file. This file is called `config.json` if you created it with the wizard, and might be called `amazon-cloudwatch-agent.json` if you created it manually.

```
& "C:\Program Files\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1" -a fetch-config -m onPremise -s -c file:configuration-file-path
```

Set up the CloudWatch agent with security-enhanced Linux (SELinux)

If your system has security-enhanced Linux (SELinux) enabled, you must apply the appropriate security policies to ensure that the CloudWatch agent runs in a confined domain.

Prerequisites

Before you can configure SELinux for the agent, check the following **prerequisites**:

To complete the prerequisites for using the CloudWatch agent with SELinux

1. If you haven't done so, install the following SELinux policy development packages:

```
sudo yum update
sudo yum install -y selinux-policy-devel policycoreutils-devel rpm-build git
```

2. Run the following command to check your system's SELinux status:

```
sestatus
```

Example output:

```
SELinux status:                 enabled
SELinuxfs mount:                /sys/fs/selinux
SELinux root directory:         /etc/selinux
Loaded policy name:             targeted
Current mode:                   permissive
Mode from config file:          permissive
Policy MLS status:              enabled
Policy deny_unknown status:     allowed
Memory protection checking:     actual (secure)
Max kernel policy version:      33
```

If you find that SELinux is currently disabled, do the following:

a. Open the SELinux file by entering the following command:

```
sudo vi /etc/selinux/config
```

b. Set the `SELINUX` parameter to either `permissive` or `enforcing`. For example:

```
SELINUX=enforcing
```

c. Save the file and reboot the system to apply the changes.

```
sudo reboot
```

3. Ensure that the CloudWatch agent is running as a `systemd` service. This is required to use it within a confined SELinux domain.

```
sudo systemctl status amazon-cloudwatch-agent
```

If the agent is correctly configured, the output should indicate that it is `active (running)` and `enabled` at startup.

Configure SELinux for the agent

After you complete the prerequisites, you can configure SELinux.

To configure SELinux for the CloudWatch agent

1. Clone the SELinux policy for the CloudWatch agent by entering the following command:

```
git clone https://github.com/aws/amazon-cloudwatch-agent-selinux.git
```

2. Navigate to the cloned repository and then update the script permissions by entering the following commands:

```
cd amazon-cloudwatch-agent-selinux
chmod +x amazon_cloudwatch_agent.sh
```

3. Use `sudo` to run the SELinux policy installation script by entering the following command. During execution, the script prompts you to enter `y` or `n` to allow automatic restart. This restart ensures that the agent transitions into the correct SELinux domain.

```
sudo ./amazon_cloudwatch_agent.sh
```

4. If the CloudWatch agent hasn't been restarted yet, restart it to ensure that it transitions to the correct SELinux domain:

```
sudo systemctl restart amazon-cloudwatch-agent
```

5. Verify that CloudWatch Agent is running in the confined domain by entering the following command:

```
ps -efZ | grep amazon-cloudwatch-agent
```

   If the agent is correctly confined, the output should indicate a SELinux-confined domain instead of `unconfined_service_t`.

   The following is an example of output when the agent is correctly confined.

```
system_u:system_r:confined_t:s0 root 1234 1 0 12:00 ? 00:00:10 /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent
```

After SELinux is configured, you can proceed to configure the agent to collect metrics, logs, and traces. For more information, see Manually create or edit the CloudWatch agent configuration file.

CloudWatch agent credentials preference

This section outlines the credentials provider chain the CloudWatch agent uses to obtain credentials when communicating with other Amazon services and APIs. The ordering is as follows:

> **Note**
>
> The preferences listed in numbers two through five are the same preference order as defined in the Amazon SDK. For more information, see Specifying Credentials in the SDK documentation.

1. Shared config and credentials files as defined in the CloudWatch agent's `common-config.toml` file. For more information, see (Optional) Modify the common configuration for proxy or Region information.
2. Amazon SDK environment variables

   > **Important**
   >
   > On Linux, if you run the CloudWatch agent using the `amazon-cloudwatch-agent-ctl` script, the script starts the agent as a `systemd` service. In this case, environment variables such as `HOME`, `AWS_ACCESS_KEY_ID`, and `AWS_SECRET_ACCESS_KEY` are not accessible by the agent.

3. Shared configuration and credentials files found in `$HOME/%USERPROFILE%`

   > **Note**
   >
   > The CloudWatch agent looks for `.aws/credentials` in `$HOME` for Linux and MacOS and looks in `%USERPROFILE%` for Windows. Unlike the Amazon SDK, the CloudWatch agent does not have fallback methods to determine the home directory if the environment variables are inaccessible. This difference in behavior is to maintain backwards compatibility with earlier implementations of the Amazon SDK.
   >
   > Furthermore, unlike with the shared credentials found in `common-config.toml`, if the Amazon SDK-derived shared credentials expire and are rotated, the renewed credentials

are not automatically picked up by the CloudWatch agent and require a restart of the agent to do so.

4. An Amazon Identity and Access Management role for tasks if an application is present that uses an Amazon Elastic Container Service task definition or a RunTask API operation.
5. An instance profile attached to an Amazon EC2 instance.

As a best practice, we recommend that you specify credentials in the following order when you use the CloudWatch agent.

1. Use IAM roles for tasks if your application uses an Amazon Elastic Container Service task definition or a RunTask API operation.
2. Use IAM roles if your application runs on an Amazon EC2 instance.
3. Use the CloudWatch agent `common-config.toml` file to specify the credentials file. This credentials file is the same one used by other Amazon SDKs and the Amazon CLI. If you're already using a shared credentials file, you can also use it for this purpose. If you provide it by using the CloudWatch agent's `common-config.toml` file, you ensure that the agent will consume rotated credentials when they expire and get replaced without requiring you to restart the agent.
4. Use environment variables. Setting environment variables is useful if you're doing development work on a computer other than an Amazon EC2 instance.

Note

If you send telemetry to a different account as explained in [Sending metrics, logs, and traces to a different account](), the CloudWatch agent uses the credentials provider chain described in this section to obtain the initial set of credentials. It then uses those credentials when assuming the IAM role specified by `role_arn` in the CloudWatch agent configuration file.

Verifying the signature of the CloudWatch agent package

GPG signature files are included for CloudWatch agent packages on Linux servers. You can use a public key to verify the agent download file is original and unmodified.

For Windows Server, you can use the MSI to verify the signature. For macOS computers, the signature is included in the agent download package.

To find the correct signature file, use the following table. For each architecture and operating system, you can see a general link and links for each Region. For example, for Amazon Linux 2023 and Amazon Linux 2 and the x86-64 architecture, three of the valid links are:

- `https://amazoncloudwatch-agent.s3.amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm.sig`
- `https://amazoncloudwatch-agent-us-east-1.s3.us-east-1.amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm`
- `https://amazoncloudwatch-agent-eu-central-1.s3.eu-central-1.amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm`

> **Note**
>
> To download the CloudWatch agent, your connection must use TLS 1.2 or later.

To verify the CloudWatch agent package on a Linux server

1. Download the public key.
2. Import the public key into your keyring.

```
shell$  gpg --import amazon-cloudwatch-agent.gpg
gpg: key 3B789C72: public key "Amazon CloudWatch Agent" imported
gpg: Total number processed: 1
gpg: imported: 1 (RSA: 1)
```

   Make a note of the key value, as you need it in the next step. In the preceding example, the key value is 3B789C72.

3. Verify the fingerprint by running the following command, replacing *key-value* with the value from the preceding step:

```
shell$  gpg --fingerprint key-value
pub   2048R/3B789C72 2017-11-14
      Key fingerprint = 9376 16F3 450B 7D80 6CBD  9725 D581 6730 3B78 9C72
```

```
uid                     Amazon CloudWatch Agent
```

The fingerprint string should be equal to the following:

`9376 16F3 450B 7D80 6CBD 9725 D581 6730 3B78 9C72`

If the fingerprint string doesn't match, don't install the agent. Contact Amazon Web Services.

After you have verified the fingerprint, you can use it to verify the signature of the CloudWatch agent package.

4. Download the package signature file using **wget**. To determine the correct signature file, see the preceding table.

```
wget Signature File Link
```

5. To verify the signature, run **gpg --verify**.

```
shell$ gpg --verify signature-filename agent-download-filename
gpg: Signature made Wed 29 Nov 2017 03:00:59 PM PST using RSA key ID 3B789C72
gpg: Good signature from "Amazon CloudWatch Agent"
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: 9376 16F3 450B 7D80 6CBD  9725 D581 6730 3B78 9C72
```

If the output includes the phrase `BAD signature`, check whether you performed the procedure correctly. If you continue to get this response, contact Amazon Web Services and avoid using the downloaded file.

Note the warning about trust. A key is trusted only if you or someone who you trust has signed it. This doesn't mean that the signature is invalid, only that you have not verified the public key.

Create the CloudWatch agent configuration file

Before running the CloudWatch agent on any servers, you must create one or more CloudWatch agent configuration files.

The agent configuration file is a JSON file that specifies the metrics, logs, and traces that the agent is to collect, including custom metrics. You can create it by using the wizard or by creating it yourself from scratch. You could also use the wizard to initially create the configuration file and then modify it manually. If you create or modify the file manually, the process is more complex, but you have more control over the metrics collected and can specify metrics not available through the wizard.

Any time you change the agent configuration file, you must then restart the agent to have the changes take effect. To restart the agent, follow the instructions in [(Optional) Modify the common configuration and named profile for CloudWatch agent](#).

After you have created a configuration file, you can save it manually as a JSON file and then use this file when installing the agent on your servers. Alternatively, you can store it in Systems Manager Parameter Store if you're going to use Systems Manager when you install the agent on servers.

The CloudWatch agent supports using multiple configuration files. For more information, see [Multiple CloudWatch agent configuration files](#).

Metrics, logs, and traces collected by the CloudWatch agent incur charges. For more information about pricing, see [Amazon CloudWatch Pricing](#).

Contents

Create the CloudWatch agent configuration file with the wizard

The agent configuration file wizard, `amazon-cloudwatch-agent-config-wizard`, asks a series of questions to help you configure the CloudWatch agent for your needs. This section describes the credentials required for the configuration file. It describes how to run the CloudWatch agent configuration wizard. It also describes the metrics that are predefined in the wizard.

Required credentials

The wizard can autodetect the credentials and Amazon Region to use if you have the Amazon credentials and configuration files in place before you start the wizard. For more information about these files, see [Configuration and Credential Files](#) in the *Amazon Systems Manager User Guide*.

In the Amazon credentials file, the wizard checks for default credentials and also looks for an `AmazonCloudWatchAgent` section such as the following:

```
[AmazonCloudWatchAgent]
aws_access_key_id = my_access_key
aws_secret_access_key = my_secret_key
```

The wizard displays the default credentials, the credentials from the `AmazonCloudWatchAgent`, and an `Others` option. You can select which credentials to use. If you choose `Others`, you can input credentials.

In the Amazon configuration file, you can specify the Region that the agent sends metrics to if it's different than the `[default]` section. The default is to publish the metrics to the Region where the Amazon EC2 instance is located. If the metrics should be published to a different Region, specify the Region here. In the following example, the metrics are published to the `us-west-1` Region.

```
[AmazonCloudWatchAgent]
region = us-west-1
```

Run the CloudWatch agent configuration wizard

To create the CloudWatch agent configuration file

1. Start the CloudWatch agent configuration wizard by entering the following:

   ```
   sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-config-wizard
   ```

   On a server running Windows Server, run the following commands to launch the wizard:

   ```
   cd "C:\Program Files\Amazon\AmazonCloudWatchAgent"
   ```

   ```
   .\amazon-cloudwatch-agent-config-wizard.exe
   ```

2. Answer the questions to customize the configuration file for your server.
3. If you're storing the configuration file locally, the configuration file `config.json` is stored in `/opt/aws/amazon-cloudwatch-agent/bin/` on Linux servers, and is stored in `C:\Program Files\Amazon\AmazonCloudWatchAgent` on Windows Server. You can then copy this file to other servers where you want to install the agent.

If you're going to use Systems Manager to install and configure the agent, be sure to answer **Yes** when prompted whether to store the file in Systems Manager Parameter Store. You can also choose to store the file in Parameter Store even if you aren't using the SSM Agent to install the CloudWatch agent. To be able to store the file in Parameter Store, you must use an IAM role with sufficient permissions. For more information, see [Create IAM roles and users for use with the CloudWatch agent](#).

CloudWatch agent predefined metric sets

The wizard is configured with predefined sets of metrics, with different detail levels. These sets of metrics are shown in the following tables. For more information about these metrics, see [Metrics collected by the CloudWatch agent](#).

> **Note**
>
> Parameter Store supports parameters in Standard and Advanced tiers. These parameter tiers are not related to the Basic, Standard, and Advanced levels of metric details that are described in these tables.

Amazon EC2 instances running Linux

Detail level	Metrics included
Basic	**Mem:** mem_used_percent **Disk:** disk_used_percent The `disk` metrics such as `disk_used_percent` have a dimension for `Partition`, which means that the number of custom metrics generated is dependent on the number of partitions associated with your instance. The number of disk partitions you have depends on which AMI you are using and the number of Amazon EBS volumes you attach to the server.
Standard	**CPU:** `cpu_usage_idle`, `cpu_usage_iowait`, `cpu_usage_user`, `cpu_usage_system` **Disk:** `disk_used_percent`, `disk_inodes_free`

Detail level	Metrics included
	Diskio: diskio_io_time **Mem:** mem_used_percent **Swap:** swap_used_percent
Advanced	**CPU:** cpu_usage_idle , cpu_usage_iowait , cpu_usage_user , cpu_usage_system **Disk:** disk_used_percent , disk_inodes_free **Diskio:** diskio_io_time , diskio_write_bytes , diskio_read_bytes , diskio_writes , diskio_reads **Mem:** mem_used_percent **Netstat:** netstat_tcp_established , netstat_tcp_time_wait **Swap:** swap_used_percent

On-premises servers running Linux

Detail level	Metrics included
Basic	**Disk:** disk_used_percent **Diskio:** diskio_write_bytes , diskio_read_bytes , diskio_writes , diskio_reads **Mem:** mem_used_percent **Net:** net_bytes_sent , net_bytes_recv , net_packets_sent , net_packets_recv **Swap:** swap_used_percent
Standard	**CPU:** cpu_usage_idle , cpu_usage_iowait **Disk:** disk_used_percent , disk_inodes_free

Detail level	Metrics included
	Diskio: diskio_io_time , diskio_write_bytes , diskio_read_bytes , diskio_writes , diskio_reads **Mem:** mem_used_percent **Net:** net_bytes_sent , net_bytes_recv , net_packets_sent , net_packets_recv **Swap:** swap_used_percent
Advanced	**CPU:** cpu_usage_guest , cpu_usage_idle , cpu_usage_iowait , cpu_usage_steal , cpu_usage_user , cpu_usage_system **Disk:** disk_used_percent , disk_inodes_free **Diskio:** diskio_io_time , diskio_write_bytes , diskio_read_bytes , diskio_writes , diskio_reads **Mem:** mem_used_percent **Net:** net_bytes_sent , net_bytes_recv , net_packets_sent , net_packets_recv **Netstat:** netstat_tcp_established , netstat_tcp_time_wait **Swap:** swap_used_percent

Amazon EC2 instances running Windows Server

> **Note**
>
> The metric names listed in this table display how the metric appears when viewed in the console. The actual metric name might not include the first word. For example, the actual metric name for `LogicalDisk % Free Space` is just `% Free Space`.

Detail level	Metrics included
Basic	**Memory:** `Memory % Committed Bytes In Use` **LogicalDisk:** `LogicalDisk % Free Space`
Standard	**Memory:** `Memory % Committed Bytes In Use` **Paging:** `Paging File % Usage` **Processor:** `Processor % Idle Time`, `Processor % Interrupt Time`, `Processor % User Time` **PhysicalDisk:** `PhysicalDisk % Disk Time` **LogicalDisk:** `LogicalDisk % Free Space`
Advanced	**Memory:** `Memory % Committed Bytes In Use` **Paging:** `Paging File % Usage` **Processor:** `Processor % Idle Time`, `Processor % Interrupt Time`, `Processor % User Time` **LogicalDisk:** `LogicalDisk % Free Space` **PhysicalDisk:** `PhysicalDisk % Disk Time` , `PhysicalDisk Disk Write Bytes/sec` , `PhysicalDisk Disk Read Bytes/sec` , `PhysicalDisk Disk Writes/sec` , `PhysicalDisk Disk Reads/sec` **TCP:** `TCPv4 Connections Established` , `TCPv6 Connections Established`

On-premises server running Windows Server

Note

The metric names listed in this table display how the metric appears when viewed in the console. The actual metric name might not include the first word. For example, the actual metric name for `LogicalDisk % Free Space` is just `% Free Space`.

Detail level	Metrics included
Basic	**Paging:** `Paging File % Usage` **Processor:** `Processor % Processor Time` **LogicalDisk:**`LogicalDisk % Free Space` **PhysicalDisk:** `PhysicalDisk Disk Write Bytes/sec` , `PhysicalDisk Disk Read Bytes/sec` , `PhysicalDisk Disk Writes/sec` , `PhysicalDisk Disk Reads/sec` **Memory:** `Memory % Committed Bytes In Use` **Network Interface:** `Network Interface Bytes Sent/sec`, `Network Interface Bytes Received/sec` , `Network Interface Packets Sent/sec`, `Network Interface Packets Received/sec`
Standard	**Paging:** `Paging File % Usage` **Processor:** `Processor % Processor Time`, `Processor % Idle Time`, `Processor % Interrupt Time` **LogicalDisk:** `LogicalDisk % Free Space` **PhysicalDisk:** `PhysicalDisk % Disk Time` , `PhysicalDisk Disk Write Bytes/sec` , `PhysicalDisk Disk Read Bytes/sec` , `PhysicalDisk Disk Writes/sec` , `PhysicalDisk Disk Reads/sec` **Memory:** `Memory % Committed Bytes In Use`

Detail level	Metrics included
	Network Interface: `Network Interface Bytes Sent/sec`, `Network Interface Bytes Received/sec` , `Network Interface Packets Sent/sec`, `Network Interface Packets Received/sec`
Advanced	**Paging:** `Paging File % Usage` **Processor:** `Processor % Processor Time`, `Processor % Idle Time`, `Processor % Interrupt Time`, `Processor % User Time` **LogicalDisk:** `LogicalDisk % Free Space` **PhysicalDisk:** `PhysicalDisk % Disk Time` , `PhysicalDisk Disk Write Bytes/sec` , `PhysicalDisk Disk Read Bytes/sec` , `PhysicalDisk Disk Writes/sec` , `PhysicalDisk Disk Reads/sec` **Memory:** `Memory % Committed Bytes In Use` **Network Interface:** `Network Interface Bytes Sent/sec`, `Network Interface Bytes Received/sec` , `Network Interface Packets Sent/sec`, `Network Interface Packets Received/sec` **TCP:** `TCPv4 Connections Established` , `TCPv6 Connections Established`

Manually create or edit the CloudWatch agent configuration file

The CloudWatch agent configuration file is a JSON file with four sections: `agent`, `metrics`, `logs`, and `traces`.

- The `agent` section includes fields for the overall configuration of the agent.
- The `metrics` section specifies the custom metrics for collection and publishing to CloudWatch. If you're using the agent only to collect logs, you can omit the `metrics` section from the file.
- The `logs` section specifies what log files are published to CloudWatch Logs. This can include events from the Windows Event Log if the server runs Windows Server.
- The `traces` section specifies the sources for traces that are collected and sent to Amazon X-Ray.

This section explains the structure and fields of the CloudWatch agent configuration file. You can view the schema definition for this configuration file. The schema definition is located at *installation-directory*/doc/amazon-cloudwatch-agent-schema.json on Linux servers and at *installation-directory*/amazon-cloudwatch-agent-schema.json on servers running Windows Server.

If you create or edit the agent configuration file manually, you can give it any name. For simplicity in troubleshooting, we recommend that you name it /opt/aws/amazon-cloudwatch-agent/etc/cloudwatch-agent.json on a Linux server and $Env:ProgramData\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent.json on servers running Windows Server. After you have created the file, you can copy it to other servers where you want to install the agent.

When the agent is started, it creates a copy of each configuration file in /opt/aws/amazon-cloudwatch/etc/amazon-cloudwatch-agent.d directory, with the filename prefixed with either file_ (for local file sources) or ssm_ (for Systems Manager parameter store sources) to indicate the configuration origin.

> **Note**
>
> Metrics, logs, and traces collected by the CloudWatch agent incur charges. For more information about pricing, see Amazon CloudWatch Pricing.

CloudWatch agent configuration file: Agent section

The agent section can include the following fields. The wizard doesn't create an agent section. Instead, the wizard omits it and uses the default values for all fields in this section.

- metrics_collection_interval – Optional. Specifies how often all metrics specified in this configuration file are to be collected. You can override this value for specific types of metrics.

  This value is specified in seconds. For example, specifying 10 causes metrics to be collected every 10 seconds, and setting it to 300 specifies metrics to be collected every 5 minutes.

  If you set this value below 60 seconds, each metric is collected as a high-resolution metric. For more information about high-resolution metrics, see High-resolution metrics.

  The default value is 60.

- `region` – Specifies the Region to use for the CloudWatch endpoint when an Amazon EC2 instance is being monitored. The metrics collected are sent to this Region, such as `us-west-1`. If you omit this field, the agent sends metrics to the Region where the Amazon EC2 instance is located.

  If you are monitoring an on-premises server, this field isn't used, and the agent reads the Region from the `AmazonCloudWatchAgent` profile of the Amazon configuration file.
- `credentials` – Specifies an IAM role to use when sending metrics, logs, and traces to a different Amazon account. If specified, this field contains one parameter, `role_arn`.
  - `role_arn` – Specifies the Amazon Resource Name (ARN) of an IAM role to use for authentication when sending metrics, logs, and traces to a different Amazon account. For more information, see Sending metrics, logs, and traces to a different account.
- `debug` – Optional. Specifies running the CloudWatch agent with debug log messages. The default value is `false`.
- `aws_sdk_log_level` – Optional. Supported only in versions 1.247350.0 and later of the CloudWatch agent.

  You can specify this field to have the agent perform logging for Amazon SDK endpoints. The value for this field can include one or more of the following options. Separate multiple options with the `|` character.
  - `LogDebug`
  - `LogDebugWithSigning`
  - `LogDebugWithHTTPBody`
  - `LogDebugRequestRetries`
  - `LogDebugWithEventStreamBody`

  For more information about these options, see LogLevelType.
- `logfile` – Specifies the location where the CloudWatch agent writes log messages. If you specify an empty string, the log goes to stderr. If you don't specify this option, the default locations are the following:
  - Linux: `/opt/aws/amazon-cloudwatch-agent/logs/amazon-cloudwatch-agent.log`
  - Windows Server: `c:\\ProgramData\\Amazon\\CloudWatchAgent\\Logs\\amazon-cloudwatch-agent.log`

The CloudWatch agent automatically rotates the log file that it creates. A log file is rotated out when it reaches 100 MB in size. The agent keeps the rotated log files for up to seven days, and it keeps as many as five backup log files that have been rotated out. Backup log files have a timestamp appended to their filename. The timestamp shows the date and time that the file was rotated out: for example, `amazon-cloudwatch-agent-2018-06-08T21-01-50.247.log.gz`.

- `omit_hostname` – Optional. By default, the hostname is published as a dimension of metrics that are collected by the agent, unless you are using the `append_dimensions` field in the `metrics` section. Set `omit_hostname` to `true` to prevent the hostname from being published as a dimension even if you are not using `append_dimensions`. The default value is `false`.
- `run_as_user` – Optional. Specifies a user to use to run the CloudWatch agent. If you don't specify this parameter, the root user is used. This option is valid only on Linux servers.

  If you specify this option, the user must exist before you start the CloudWatch agent. For more information, see Running the CloudWatch agent as a different user.
- `user_agent` – Optional. Specifies the `user-agent` string that is used by the CloudWatch agent when it makes API calls to the CloudWatch backend. The default value is a string consisting of the agent version, the version of the Go programming language that was used to compile the agent, the runtime operating system and architecture, the build time, and the plugins enabled.
- `usage_data` – Optional. By default, the CloudWatch agent sends health and performance data about itself to CloudWatch whenever it publishes metrics or logs to CloudWatch. This data incurs no costs to you. You can prevent the agent from sending this data by specifying `false` for `usage_data`. If you omit this parameter, the default of `true` is used and the agent sends the health and performance data.

  If you set this value to `false`, you must stop and restart the agent for it to take effect.
- `service.name` – Optional. Specifies the service name to be used to populate the entity for finding related telemetry.
- `deployment.environment` – Optional. Specifies the environment name to be used to populate the entity for finding related telemetry.

The following is an example of an `agent` section.

```
"agent": {
   "metrics_collection_interval": 60,
```

```
    "region": "us-west-1",
    "logfile": "/opt/aws/amazon-cloudwatch-agent/logs/amazon-cloudwatch-agent.log",
    "debug": false,
    "run_as_user": "cwagent"
  }
```

CloudWatch agent configuration file: Metrics section

Fields common to Linux and Windows

On servers running either Linux or Windows Server, the `metrics` section includes the following fields:

- `namespace` – Optional. The namespace to use for the metrics collected by the agent. The default value is `CWAgent`. The maximum length is 255 characters. The following is an example:

  ```
  {
    "metrics": {
      "namespace": "Development/Product1Metrics",
     ......
     },
  }
  ```

- `append_dimensions` – Optional. Adds Amazon EC2 metric dimensions to all metrics collected by the agent. This also causes the agent to not publish the hostname as a dimension.

  The only supported key-value pairs for `append_dimensions` are shown in the following list. Any other key-value pairs are ignored. The agent supports these key-value pairs exactly as they are shown in the following list. You can't change the key values to publish different dimension names for them.

  - `"ImageId":"${aws:ImageId}"` sets the instance's AMI ID as the value of the `ImageId` dimension.
  - `"InstanceId":"${aws:InstanceId}"` sets the instance's instance ID as the value of the `InstanceId` dimension.
  - `"InstanceType":"${aws:InstanceType}"` sets the instance's instance type as the value of the `InstanceType` dimension.
  - `"AutoScalingGroupName":"${aws:AutoScalingGroupName}"` sets the instance's Auto Scaling group name as the value of the `AutoScalingGroupName` dimension.

If you want to append dimensions to metrics with arbitrary key-value pairs, use the `append_dimensions` parameter in the field for that particular type of metric.

If you specify a value that depends on Amazon EC2 metadata and you use proxies, you must make sure that the server can access the endpoint for Amazon EC2. For more information about these endpoints, see Amazon Elastic Compute Cloud (Amazon EC2) in the *Amazon Web Services General Reference*.

- `aggregation_dimensions` – Optional. Specifies the dimensions that collected metrics are to be aggregated on. For example, if you roll up metrics on the `AutoScalingGroupName` dimension, the metrics from all instances in each Auto Scaling group are aggregated and can be viewed as a whole.

  You can roll up metrics along single or multiple dimensions. For example, specifying `[["InstanceId"], ["InstanceType"], ["InstanceId","InstanceType"]]` aggregates metrics for instance ID singly, instance type singly, and for the combination of the two dimensions.

  You can also specify `[]` to roll up all metrics into one collection, disregarding all dimensions.

- `endpoint_override` – Specifies a FIPS endpoint or private link to use as the endpoint where the agent sends metrics. Specifying this and setting a private link enables you to send the metrics to an Amazon VPC endpoint. For more information, see What Is Amazon VPC?.

  The value of `endpoint_override` must be a string that is a URL.

  For example, the following part of the metrics section of the configuration file sets the agent to use a VPC Endpoint when sending metrics.

```
{
  "metrics": {
    "endpoint_override": "vpce-XXXXXXXXXXXXXXXXXXXXXXXXX.monitoring.us-
east-1.vpce.amazonaws.com",
   ......
   },
}
```

- `metrics_collected` – Required. Specifies which metrics are to be collected, including custom metrics collected through `StatsD` or `collectd`. This section includes several subsections.

The contents of the `metrics_collected` section depend on whether this configuration file is for a server running Linux or Windows Server.

- `metrics_destinations` – Optional. Specifies one or more destinations for all metrics defined in `metrics_collected`. If specified here, it overrides the default destination of `cloudwatch`.
  - `cloudwatch` – Amazon CloudWatch.
  - amp – Amazon Managed Service for Prometheus.
    - `workspace_id` – The ID corresponding to the Amazon Managed Service for Prometheus workspace.

```
{
  "metrics": {
    "metrics_destinations": {
      "cloudwatch": {},
      "amp": {
        "workspace_id": "ws-abcd1234-ef56-7890-ab12-example"
      }
    }
  }
}
```

- `force_flush_interval` – Specifies in seconds the maximum amount of time that metrics remain in the memory buffer before being sent to the server. No matter the setting for this, if the size of the metrics in the buffer reaches 1 MB or 1000 different metrics, the metrics are immediately sent to the server.

  The default value is 60.
- `credentials` – Specifies an IAM role to use when sending metrics to a different account. If specified, this field contains one parameter, `role_arn`.
  - `role_arn` – Specifies the ARN of an IAM role to use for authentication when sending metrics to a different account. For more information, see [Sending metrics, logs, and traces to a different account](). If specified here, this value overrides the `role_arn` specified in the `agent` section of the configuration file, if any.
  - `service.name` – Optional. Specifies the service name to be used to populate the entity for [finding related telemetry]().
  - `deployment.environment` – Optional. Specifies the environment name to be used to populate the entity for [finding related telemetry]().

Linux section

On servers running Linux, the `metrics_collected` section of the configuration file can also contain the following fields.

Many of these fields can include a `measurement` sections that lists the metrics you want to collect for that resource. These `measurement` sections can either specify the complete metric name such as `swap_used`, or just the part of the metric name that will be appended to the type of resource. For example, specifying `reads` in the `measurement` section of the `diskio` section causes the `diskio_reads` metric to be collected.

- `collectd` – Optional. Specifies that you want to retrieve custom metrics using the `collectd` protocol. You use `collectd` software to send the metrics to the CloudWatch agent. For more information about the configuration options available for collectd, see [Retrieve custom metrics with collectd](#).
- `cpu` – Optional. Specifies that CPU metrics are to be collected. This section is valid only for Linux instances. You must include at least one of the `resources` and `totalcpu` fields for any CPU metrics to be collected. This section can include the following fields:
  - `drop_original_metrics` – Optional. If you are using the `aggregation_dimensions` field in the `metrics` section to roll up metrics into aggregated results, then by default the agent sends both the aggregated metrics and the original metrics that are separated for each value of the dimension. If you don't want the original metrics to be sent to CloudWatch, you can specify this parameter with a list of metrics. The metrics specified along with this parameter don't have their metrics by dimension reported to CloudWatch. Instead, only the aggregated metrics are reported. This reduces the number of metrics that the agent collects, reducing your costs.
  - `resources` – Optional. Specify this field with a value of `*` to cause per-cpu metrics to be collected. The only allowed value is `*`.
  - `totalcpu` – Optional. Specifies whether to report cpu metrics aggregated across all cpu cores. The default is true.
  - `measurement` – Specifies the array of cpu metrics to be collected. Possible values are `time_active`, `time_guest`, `time_guest_nice`, `time_idle`, `time_iowait`, `time_irq`, `time_nice`, `time_softirq`, `time_steal`, `time_system`, `time_user`, `usage_active`, `usage_guest`, `usage_guest_nice`, `usage_idle`, `usage_iowait`, `usage_irq`, `usage_nice`, `usage_softirq`, `usage_steal`, `usage_system`, and `usage_user`. This field is required if you include `cpu`.

By default, the unit for `cpu_usage_*` metrics is `Percent`, and `cpu_time_*` metrics don't have a unit.

Within the entry for each individual metric, you might optionally specify one or both of the following:

- `rename` – Specifies a different name for this metric.
- `unit` – Specifies the unit to use for this metric, overriding the default unit of `None` for the metric. The unit that you specify must be a valid CloudWatch metric unit, as listed in the `Unit` description in [MetricDatum](https://docs.aws.amazon.com/AmazonCloudWatch/latest/APIReference/API_MetricDatum.html).

- `metrics_collection_interval` – Optional. Specifies how often to collect the cpu metrics, overriding the global `metrics_collection_interval` specified in the `agent` section of the configuration file.

  This value is specified in seconds. For example, specifying 10 causes metrics to be collected every 10 seconds, and setting it to 300 specifies metrics to be collected every 5 minutes.

  If you set this value below 60 seconds, each metric is collected as a high-resolution metric. For more information about high-resolution metrics, see [High-resolution metrics](https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/publishingMetrics.html#high-resolution-metrics).
- `append_dimensions` – Optional. Additional dimensions to use for only the cpu metrics. If you specify this field, it's used in addition to dimensions specified in the global `append_dimensions` field that is used for all types of metrics that the agent collects.

- `disk` – Optional. Specifies that disk metrics are to be collected. Collects metrics only for mounted volumes. This section is valid only for Linux instances. This section can include the following fields:

  - `drop_original_metrics` – Optional. If you are using the `aggregation_dimensions` field in the `metrics` section to roll up metrics into aggregated results, then by default the agent sends both the aggregated metrics and the original metrics that are separated for each value of the dimension. If you don't want the original metrics to be sent to CloudWatch, you can specify this parameter with a list of metrics. The metrics specified along with this parameter don't have their metrics by dimension reported to CloudWatch. Instead, only the aggregated metrics are reported. This reduces the number of metrics that the agent collects, reducing your costs.
  - `resources` – Optional. Specifies an array of disk mount points. This field limits CloudWatch to collect metrics from only the listed mount points. You can specify `*` as the value to collect metrics from all mount points. The default value is to collect metrics from all mount points.

- `measurement` – Specifies the array of disk metrics to be collected. Possible values are `free`, `total`, `used`, `used_percent`, `inodes_free`, `inodes_used`, and `inodes_total`. This field is required if you include `disk`.

> **Note**
>
> The `disk` metrics have a dimension for `Partition`, which means that the number of custom metrics generated is dependent on the number of partitions associated with your instance. The number of disk partitions you have depends on which AMI you are using and the number of Amazon EBS volumes you attach to the server.

  To see the default units for each `disk` metric, see [Metrics collected by the CloudWatch agent on Linux and macOS instances](#).

  Within the entry for each individual metric, you might optionally specify one or both of the following:

  - `rename` – Specifies a different name for this metric.
  - `unit` – Specifies the unit to use for this metric, overriding the default unit of None of None for the metric. The unit that you specify must be a valid CloudWatch metric unit, as listed in the `Unit` description in [MetricDatum](#).
- `ignore_file_system_types` – Specifies file system types to exclude when collecting disk metrics. Valid values include `sysfs`, `devtmpfs`, and so on.
- `drop_device` – Setting this to `true` causes `Device` to not be included as a dimension for disk metrics.

  Preventing `Device` from being used as a dimension can be useful on instances that use the Nitro system because on those instances the device names change for each disk mount when the instance is rebooted. This can cause inconsistent data in your metrics and cause alarms based on these metrics to go to `INSUFFICIENT DATA` state.

  The default is `false`.
- `metrics_collection_interval` – Optional. Specifies how often to collect the disk metrics, overriding the global `metrics_collection_interval` specified in the `agent` section of the configuration file.

  This value is specified in seconds.

If you set this value below 60 seconds, each metric is collected as a high-resolution metric. For more information, see High-resolution metrics.

- `append_dimensions` – Optional. Specify key-value pairs to use as additional dimensions for only the disk metrics. If you specify this field, it is used in addition to dimensions specified in the `append_dimensions` field that is used for all types of metrics collected by the agent.

  One key-value pair that you can use is the following. You can also specify other custom key-value pairs.

  - `"VolumeId":"${aws:VolumeId}"` adds a `VolumeId` dimension to your block device disk metrics. For Amazon EBS volumes, this will be the Amazon EBS Volume ID. For EC2 instance store, this will be the device serial. Using this requires the `drop_device` parameter to be set to `false`.

- `diskio` – Optional. Specifies that disk i/o metrics are to be collected. This section is valid only for Linux instances. This section can include the following fields:

  - `drop_original_metrics` – Optional. If you are using the `aggregation_dimensions` field in the `metrics` section to roll up metrics into aggregated results, then by default the agent sends both the aggregated metrics and the original metrics that are separated for each value of the dimension. If you don't want the original metrics to be sent to CloudWatch, you can specify this parameter with a list of metrics. The metrics specified along with this parameter don't have their metrics by dimension reported to CloudWatch. Instead, only the aggregated metrics are reported. This reduces the number of metrics that the agent collects, reducing your costs.

  - `resources` – Optional. If you specify an array of devices, CloudWatch collects metrics from only those devices. Otherwise, metrics for all devices are collected. You can also specify * as the value to collect metrics from all devices.

  - `measurement` – Specifies the array of diskio metrics to be collected. Possible values are `reads`, `writes`, `read_bytes`, `write_bytes`, `read_time`, `write_time`, `io_time`, and `iops_in_progress`. This field is required if you include `diskio`.

    Within the entry for each individual metric, you might optionally specify one or both of the following:

    - `rename` – Specifies a different name for this metric.

    - `unit` – Specifies the unit to use for this metric, overriding the default unit of None of None for the metric. The unit that you specify must be a valid CloudWatch metric unit, as listed in the `Unit` description in MetricDatum.

  - `metrics_collection_interval` – Optional. Specifies how often to collect the diskio metrics, overriding the global `metrics_collection_interval` specified in the `agent` section of the configuration file.

    This value is specified in seconds.

    If you set this value below 60 seconds, each metric is collected as a high-resolution metric. For more information about high-resolution metrics, see High-resolution metrics.
  - `append_dimensions` – Optional. Additional dimensions to use for only the diskio metrics. If you specify this field, it is used in addition to dimensions specified in the `append_dimensions` field that is used for all types of metrics collected by the agent.
- `swap` – Optional. Specifies that swap memory metrics are to be collected. This section is valid only for Linux instances. This section can include the following fields:
  - `drop_original_metrics` – Optional. If you are using the `aggregation_dimensions` field in the `metrics` section to roll up metrics into aggregated results, then by default the agent sends both the aggregated metrics and the original metrics that are separated for each value of the dimension. If you don't want the original metrics to be sent to CloudWatch, you can specify this parameter with a list of metrics. The metrics specified along with this parameter don't have their metrics by dimension reported to CloudWatch. Instead, only the aggregated metrics are reported. This reduces the number of metrics that the agent collects, reducing your costs.
  - `measurement` – Specifies the array of swap metrics to be collected. Possible values are `free`, `used`, and `used_percent`. This field is required if you include `swap`.

    To see the default units for each `swap` metric, see Metrics collected by the CloudWatch agent on Linux and macOS instances.

    Within the entry for each individual metric, you might optionally specify one or both of the following:
    - `rename` – Specifies a different name for this metric.
    - `unit` – Specifies the unit to use for this metric, overriding the default unit of None of None for the metric. The unit that you specify must be a valid CloudWatch metric unit, as listed in the `Unit` description in MetricDatum.
  - `metrics_collection_interval` – Optional. Specifies how often to collect the swap metrics, overriding the global `metrics_collection_interval` specified in the `agent` section of the configuration file.

This value is specified in seconds.

If you set this value below 60 seconds, each metric is collected as a high-resolution metric. For more information about high-resolution metrics, see High-resolution metrics.

- `append_dimensions` – Optional. Additional dimensions to use for only the swap metrics. If you specify this field, it is used in addition to dimensions specified in the global `append_dimensions` field that is used for all types of metrics collected by the agent. It's collected as a high-resolution metric.

- `mem` – Optional. Specifies that memory metrics are to be collected. This section is valid only for Linux instances. This section can include the following fields:
  - `drop_original_metrics` – Optional. If you are using the `aggregation_dimensions` field in the `metrics` section to roll up metrics into aggregated results, then by default the agent sends both the aggregated metrics and the original metrics that are separated for each value of the dimension. If you don't want the original metrics to be sent to CloudWatch, you can specify this parameter with a list of metrics. The metrics specified along with this parameter don't have their metrics by dimension reported to CloudWatch. Instead, only the aggregated metrics are reported. This reduces the number of metrics that the agent collects, reducing your costs.
  - `measurement` – Specifies the array of memory metrics to be collected. Possible values are `active`, `available`, `available_percent`, `buffered`, `cached`, `free`, `inactive`, `total`, `used`, and `used_percent`. This field is required if you include `mem`.

    To see the default units for each `mem` metric, see Metrics collected by the CloudWatch agent on Linux and macOS instances.

    Within the entry for each individual metric, you might optionally specify one or both of the following:
    - `rename` – Specifies a different name for this metric.
    - `unit` – Specifies the unit to use for this metric, overriding the default unit of `None` for the metric. The unit that you specify must be a valid CloudWatch metric unit, as listed in the `Unit` description in MetricDatum.
  - `metrics_collection_interval` – Optional. Specifies how often to collect the mem metrics, overriding the global `metrics_collection_interval` specified in the `agent` section of the configuration file.

    This value is specified in seconds.

If you set this value below 60 seconds, each metric is collected as a high-resolution metric. For more information about high-resolution metrics, see High-resolution metrics.

  - `append_dimensions` – Optional. Additional dimensions to use for only the mem metrics. If you specify this field, it's used in addition to dimensions specified in the `append_dimensions` field that is used for all types of metrics that the agent collects.
- `net` – Optional. Specifies that networking metrics are to be collected. This section is valid only for Linux instances. This section can include the following fields:
  - `drop_original_metrics` – Optional. If you are using the `aggregation_dimensions` field in the `metrics` section to roll up metrics into aggregated results, then by default the agent sends both the aggregated metrics and the original metrics that are separated for each value of the dimension. If you don't want the original metrics to be sent to CloudWatch, you can specify this parameter with a list of metrics. The metrics specified along with this parameter don't have their metrics by dimension reported to CloudWatch. Instead, only the aggregated metrics are reported. This reduces the number of metrics that the agent collects, reducing your costs.
  - `resources` – Optional. If you specify an array of network interfaces, CloudWatch collects metrics from only those interfaces. Otherwise, metrics for all devices are collected. You can also specify `*` as the value to collect metrics from all interfaces.
  - `measurement` – Specifies the array of networking metrics to be collected. Possible values are `bytes_sent`, `bytes_recv`, `drop_in`, `drop_out`, `err_in`, `err_out`, `packets_sent`, and `packets_recv`. This field is required if you include `net`.

    To see the default units for each `net` metric, see Metrics collected by the CloudWatch agent on Linux and macOS instances.

    Within the entry for each individual metric, you might optionally specify one or both of the following:
    - `rename` – Specifies a different name for this metric.
    - `unit` – Specifies the unit to use for this metric, overriding the default unit of None for the metric. The unit that you specify must be a valid CloudWatch metric unit, as listed in the `Unit` description in MetricDatum.
  - `metrics_collection_interval` – Optional. Specifies how often to collect the net metrics, overriding the global `metrics_collection_interval` specified in the `agent` section of the configuration file.

This value is specified in seconds. For example, specifying 10 causes metrics to be collected every 10 seconds, and setting it to 300 specifies metrics to be collected every 5 minutes.

If you set this value below 60 seconds, each metric is collected as a high-resolution metric. For more information about high-resolution metrics, see High-resolution metrics.

  - `append_dimensions` – Optional. Additional dimensions to use for only the net metrics. If you specify this field, it's used in addition to dimensions specified in the `append_dimensions` field that is used for all types of metrics collected by the agent.

- `netstat` – Optional. Specifies that TCP connection state and UDP connection metrics are to be collected. This section is valid only for Linux instances. This section can include the following fields:
  - `drop_original_metrics` – Optional. If you are using the `aggregation_dimensions` field in the `metrics` section to roll up metrics into aggregated results, then by default the agent sends both the aggregated metrics and the original metrics that are separated for each value of the dimension. If you don't want the original metrics to be sent to CloudWatch, you can specify this parameter with a list of metrics. The metrics specified along with this parameter don't have their metrics by dimension reported to CloudWatch. Instead, only the aggregated metrics are reported. This reduces the number of metrics that the agent collects, reducing your costs.
  - `measurement` – Specifies the array of netstat metrics to be collected. Possible values are `tcp_close`, `tcp_close_wait`, `tcp_closing`, `tcp_established`, `tcp_fin_wait1`, `tcp_fin_wait2`, `tcp_last_ack`, `tcp_listen`, `tcp_none`, `tcp_syn_sent`, `tcp_syn_recv`, `tcp_time_wait`, and `udp_socket`. This field is required if you include `netstat`.

    To see the default units for each `netstat` metric, see Metrics collected by the CloudWatch agent on Linux and macOS instances.

    Within the entry for each individual metric, you might optionally specify one or both of the following:
    - `rename` – Specifies a different name for this metric.
    - `unit` – Specifies the unit to use for this metric, overriding the default unit of `None` for the metric. The unit that you specify must be a valid CloudWatch metric unit, as listed in the `Unit` description in MetricDatum.

  - `metrics_collection_interval` – Optional. Specifies how often to collect the netstat metrics, overriding the global `metrics_collection_interval` specified in the `agent` section of the configuration file.

    This value is specified in seconds.

    If you set this value below 60 seconds, each metric is collected as a high-resolution metric. For more information about high-resolution metrics, see High-resolution metrics.
  - `append_dimensions` – Optional. Additional dimensions to use for only the netstat metrics. If you specify this field, it's used in addition to dimensions specified in the `append_dimensions` field that is used for all types of metrics collected by the agent.
- `processes` – Optional. Specifies that process metrics are to be collected. This section is valid only for Linux instances. This section can include the following fields:
  - `drop_original_metrics` – Optional. If you are using the `aggregation_dimensions` field in the `metrics` section to roll up metrics into aggregated results, then by default the agent sends both the aggregated metrics and the original metrics that are separated for each value of the dimension. If you don't want the original metrics to be sent to CloudWatch, you can specify this parameter with a list of metrics. The metrics specified along with this parameter don't have their metrics by dimension reported to CloudWatch. Instead, only the aggregated metrics are reported. This reduces the number of metrics that the agent collects, reducing your costs.
  - `measurement` – Specifies the array of processes metrics to be collected. Possible values are `blocked`, `dead`, `idle`, `paging`, `running`, `sleeping`, `stopped`, `total`, `total_threads`, `wait`, and `zombies`. This field is required if you include `processes`.

    For all `processes` metrics, the default unit is `None`.

    Within the entry for each individual metric, you might optionally specify one or both of the following:
    - `rename` – Specifies a different name for this metric.
    - `unit` – Specifies the unit to use for this metric, overriding the default unit of `None` for the metric. The unit that you specify must be a valid CloudWatch metric unit, as listed in the `Unit` description in MetricDatum.
  - `metrics_collection_interval` – Optional. Specifies how often to collect the processes metrics, overriding the global `metrics_collection_interval` specified in the `agent` section of the configuration file.

This value is specified in seconds. For example, specifying 10 causes metrics to be collected every 10 seconds, and setting it to 300 specifies metrics to be collected every 5 minutes.

If you set this value below 60 seconds, each metric is collected as a high-resolution metric. For more information, see High-resolution metrics.

- `append_dimensions` – Optional. Additional dimensions to use for only the process metrics. If you specify this field, it's used in addition to dimensions specified in the `append_dimensions` field that is used for all types of metrics collected by the agent.

- `nvidia_gpu` – Optional. Specifies that NVIDIA GPU metrics are to be collected. This section is valid only for Linux instances on hosts that are configured with a NVIDIA GPU accelerator and have the NVIDIA System Management Interface (nvidia-smi) installed.

  The NVIDIA GPU metrics that are collected are prefixed with the string `nvidia_smi_` to distinguish them from the metrics collected for other accelerator types. This section can include the following fields:

  - `drop_original_metrics` – Optional. If you are using the `aggregation_dimensions` field in the `metrics` section to roll up metrics into aggregated results, then by default the agent sends both the aggregated metrics and the original metrics that are separated for each value of the dimension. If you don't want the original metrics to be sent to CloudWatch, you can specify this parameter with a list of metrics. The metrics specified along with this parameter don't have their metrics by dimension reported to CloudWatch. Instead, only the aggregated metrics are reported. This reduces the number of metrics that the agent collects, reducing your costs.

  - `measurement` – Specifies the array of NVIDIA GPU metrics to be collected. For a list of the possible values to use here, see the **Metric** column in the table in Collect NVIDIA GPU metrics.

    Within the entry for each individual metric, you can optionally specify one or both of the following:

    - `rename` – Specifies a different name for this metric.
    - `unit` – Specifies the unit to use for this metric, overriding the default unit of `None` for the metric. The unit that you specify must be a valid CloudWatch metric unit, as listed in the `Unit` description in MetricDatum.

  - `metrics_collection_interval` – Optional. Specifies how often to collect the NVIDIA GPU metrics, overriding the global `metrics_collection_interval` specified in the `agent` section of the configuration file.

- `jmx` – Optional. Specifies that you want to retrieve Java Management Extensions (JMX) metrics from the instance. For more information about the parameters that you can use in this section and the metrics that you can collect, see Collect Java Management Extensions (JMX) metrics.
- `otlp` – Optional. Specifies that you want to collect metrics from the OpenTelemetry SDK. For more information about the fields that you can use in this section, see Collect metrics and traces with OpenTelemetry.
- `procstat` – Optional. Specifies that you want to retrieve metrics from individual processes. For more information about the configuration options available for procstat, see Collect process metrics with the procstat plugin.
- `statsd` – Optional. Specifies that you want to retrieve custom metrics using the `StatsD` protocol. The CloudWatch agent acts as a daemon for the protocol. You use any standard `StatsD` client to send the metrics to the CloudWatch agent. For more information about the configuration options available for StatsD, see Retrieve custom metrics with StatsD.
- `ethtool` – Optional. Specifies that you want to retrieve network metrics using the `ethtool` plugin. This plugin can import both the metrics collected by the standard ethtool utility, as well as network performance metrics from Amazon EC2 instances. For more information about the configuration options available for ethtool, see Collect network performance metrics.

The following is an example of a `metrics` section for a Linux server. In this example, three CPU metrics, three netstat metrics, three process metrics, and one disk metric are collected, and the agent is set up to receive additional metrics from a `collectd` client.

```
"metrics": {
    "aggregation_dimensions" : [["AutoScalingGroupName"], ["InstanceId",
 "InstanceType"],[]],
    "metrics_collected": {
      "collectd": {},
      "cpu": {
        "resources": [
          "*"
        ],
        "measurement": [
          {"name": "cpu_usage_idle", "rename": "CPU_USAGE_IDLE", "unit": "Percent"},
          {"name": "cpu_usage_nice", "unit": "Percent"},
          "cpu_usage_guest"
        ],
        "totalcpu": false,
        "drop_original_metrics": [ "cpu_usage_guest" ],
```

```
        "metrics_collection_interval": 10,
        "append_dimensions": {
          "test": "test1",
          "date": "2017-10-01"
        }
      },
      "netstat": {
        "measurement": [
          "tcp_established",
          "tcp_syn_sent",
          "tcp_close"
        ],
        "metrics_collection_interval": 60
      },
       "disk": {
        "measurement": [
          "used_percent"
        ],
        "resources": [
          "*"
        ],
        "drop_device": true
      },
      "processes": {
        "measurement": [
          "running",
          "sleeping",
          "dead"
        ]
      }
    },
    "append_dimensions": {
      "ImageId": "${aws:ImageId}",
      "InstanceId": "${aws:InstanceId}",
      "InstanceType": "${aws:InstanceType}",
      "AutoScalingGroupName": "${aws:AutoScalingGroupName}"
    }
  }
```

Windows Server

In the `metrics_collected` section for Windows Server, you can have subsections for each Windows performance object, such as `Memory`, `Processor`, and `LogicalDisk`. For information about what objects and counters are available, see Performance Counters in the Microsoft Windows documentation.

Within the subsection for each object, you specify a `measurement` array of the counters to collect. The `measurement` array is required for each object that you specify in the configuration file. You can also specify a `resources` field to name the instances to collect metrics from. You can also specify `*` for `resources` to collect separate metrics for every instance. If you omit `resources` for counters that have instances, the data for all instances is aggregated into one set. If you omit `resources` for counters that don't have instances, the counters are not collected by the CloudWatch agent. To determine whether counters have instances, you can use one of the following commands.

Powershell:

```
Get-Counter -ListSet *
```

Command line (not Powershell):

```
TypePerf.exe -q
```

Within each object section, you can also specify the following optional fields:

- `metrics_collection_interval` – Optional. Specifies how often to collect the metrics for this object, overriding the global `metrics_collection_interval` specified in the `agent` section of the configuration file.

  This value is specified in seconds. For example, specifying 10 causes metrics to be collected every 10 seconds, and setting it to 300 specifies metrics to be collected every 5 minutes.

  If you set this value below 60 seconds, each metric is collected as a high-resolution metric. For more information, see High-resolution metrics.
- `append_dimensions` – Optional. Specifies additional dimensions to use for only the metrics for this object. If you specify this field, it's used in addition to dimensions specified in the global `append_dimensions` field that is used for all types of metrics collected by the agent.
- `drop_original_metrics` – Optional. If you are using the `aggregation_dimensions` field in the `metrics` section to roll up metrics into aggregated results, then by default the agent sends both the aggregated metrics and the original metrics that are separated for each value of the dimension. If you don't want the original metrics to be sent to CloudWatch, you can specify

this parameter with a list of metrics. The metrics specified along with this parameter don't have their metrics by dimension reported to CloudWatch. Instead, only the aggregated metrics are reported. This reduces the number of metrics that the agent collects, reducing your costs.

Within each counter section, you can also specify the following optional fields:

- `rename` – Specifies a different name to be used in CloudWatch for this metric.
- `unit` – Specifies the unit to use for this metric. The unit that you specify must be a valid CloudWatch metric unit, as listed in the `Unit` description in MetricDatum.

There are other optional sections that you can include in `metrics_collected`:

- `statsd` – Enables you to retrieve custom metrics using the `StatsD` protocol. The CloudWatch agent acts as a daemon for the protocol. You use any standard `StatsD` client to send the metrics to the CloudWatch agent. For more information, see Retrieve custom metrics with StatsD.
- `procstat` – Enables you to retrieve metrics from individual processes. For more information, see Collect process metrics with the procstat plugin.
- `jmx` – Optional. Specifies that you want to retrieve Java Management Extensions (JMX) metrics from the instance. For more information about the fields that you can use in this section and the metrics that you can collect, see Collect Java Management Extensions (JMX) metrics.
- `otlp` – Optional. Specifies that you want to collect metrics from the OpenTelemetry SDK. For more information about the fields that you can use in this section, see Collect metrics and traces with OpenTelemetry.

The following is an example `metrics` section for use on Windows Server. In this example, many Windows metrics are collected, and the computer is also set to receive additional metrics from a `StatsD` client.

```
"metrics": {
    "metrics_collected": {
      "statsd": {},
      "Processor": {
        "measurement": [
          {"name": "% Idle Time", "rename": "CPU_IDLE", "unit": "Percent"},
          "% Interrupt Time",
          "% User Time",
          "% Processor Time"
```

```
        ],
        "resources": [
          "*"
        ],
        "append_dimensions": {
          "d1": "win_foo",
          "d2": "win_bar"
        }
      },
      "LogicalDisk": {
        "measurement": [
          {"name": "% Idle Time", "unit": "Percent"},
          {"name": "% Disk Read Time", "rename": "DISK_READ"},
          "% Disk Write Time"
        ],
        "resources": [
          "*"
        ]
      },
      "Memory": {
        "metrics_collection_interval": 5,
        "measurement": [
          "Available Bytes",
          "Cache Faults/sec",
          "Page Faults/sec",
          "Pages/sec"
        ],
        "append_dimensions": {
          "d3": "win_bo"
        }
      },
      "Network Interface": {
        "metrics_collection_interval": 5,
        "measurement": [
          "Bytes Received/sec",
          "Bytes Sent/sec",
          "Packets Received/sec",
          "Packets Sent/sec"
        ],
        "resources": [
          "*"
        ],
        "append_dimensions": {
          "d3": "win_bo"
```

```
        }
      },
      "System": {
        "measurement": [
          "Context Switches/sec",
          "System Calls/sec",
          "Processor Queue Length"
        ],
        "append_dimensions": {
          "d1": "win_foo",
          "d2": "win_bar"
        }
      }
    },
    "append_dimensions": {
      "ImageId": "${aws:ImageId}",
      "InstanceId": "${aws:InstanceId}",
      "InstanceType": "${aws:InstanceType}",
      "AutoScalingGroupName": "${aws:AutoScalingGroupName}"
    },
    "aggregation_dimensions" : [["ImageId"], ["InstanceId", "InstanceType"], ["d1"],[]]
    }
  }
```

CloudWatch agent configuration file: Logs section

The `logs` section includes the following fields:

- `service.name` – Optional. Specifies the service name to be used to populate the entity for finding related telemetry.
- `deployment.environment` – Optional. Specifies the environment name to be used to populate the entity for finding related telemetry.
- `logs_collected` – Required if the `logs` section is included. Specifies which log files and Windows event logs are to be collected from the server. It can include two fields, `files` and `windows_events`.
  - `files` – Specifies which regular log files the CloudWatch agent is to collect. It contains one field, `collect_list`, which further defines these files.
    - `collect_list` – Required if `files` is included. Contains an array of entries, each of which specifies one log file to collect. Each of these entries can include the following fields:

- `file_path` – Specifies the path of the log file to upload to CloudWatch Logs. Standard Unix glob matching rules are accepted, with the addition of `**` as a *super asterisk*. For example, specifying `/var/log/**.log` causes all `.log` files in the `/var/log` directory tree to be collected. For more examples, see Glob Library.

  You can also use the standard asterisk as a standard wildcard. For example, `/var/log/system.log*` matches files such as `system.log_1111`, `system.log_2222`, and so on in `/var/log`.

  Only the latest file is pushed to CloudWatch Logs based on file modification time. We recommend that you use wildcards to specify a series of files of the same type, such as `access_log.2018-06-01-01` and `access_log.2018-06-01-02`, but not multiple kinds of files, such as `access_log_80` and `access_log_443`. To specify multiple kinds of files, add another log stream entry to the agent configuration file so that each kind of log file goes to a different log stream.
- `auto_removal` – Optional. If this is `true`, the CloudWatch agent automatically deletes this log file after reading it and it has been rotated. Usually the log files are deleted after their entire contents are uploaded to CloudWatch Logs, but if the agent reaches the EOF (end of file) and also detects another newer log file that matches the same `file_path`, the agent deletes the OLD file, so you must make sure that you are done writing to the OLD file before creating the NEW file. The RUST tracing library has a known incompatibility because it will potentially create a NEW log file and then still attempt to write to the OLD log file.

  The agent only removes complete files from logs that create multiple files, such as logs that create separate files for each date. If a log continuously writes to a single file, it is not removed.

  If you already have a log file rotation or removal method in place, we recommend that you omit this field or set it to `false`.

  If you omit this field, the default value of `false` is used.
- `log_group_name` – Optional. Specifies what to use as the log group name in CloudWatch Logs.

  We recommend that you use this field to specify a log group name to prevent confusion. If you omit `log_group_name`, the value of `file_path` up to the final dot is used as the log group name. For example, if the file path is `/tmp/TestLogFile.log.2017-07-11-14`, the log group name is `/tmp/TestLogFile.log`.

If you specify a log group name, you can use `{instance_id}`, `{hostname}`, `{local_hostname}`, and `{ip_address}` as variables within the name. `{hostname}` retrieves the hostname from the EC2 metadata, and `{local_hostname}` uses the hostname from the network configuration file.

If you use these variables to create many different log groups, keep in mind the limit of 1,000,000 log groups per Region per account.

Allowed characters include a–z, A–Z, 0–9, '_' (underscore), '-' (hyphen), '/' (forward slash), and '.' (period).

- `log_group_class` – Optional. Specifies which log group class to use for the new log group. For more information about log group classes, see [Log classes](#).

  Valid values are `STANDARD` and `INFREQUENT_ACCESS`. If you omit this field, the default of `STANDARD` is used.

  > **Important**
  >
  > After a log group is created, its class can't be changed.

- `log_stream_name` – Optional. Specifies what to use as the log stream name in CloudWatch Logs. As part of the name, you can use `{instance_id}`, `{hostname}`, `{local_hostname}`, and `{ip_address}` as variables within the name. `{hostname}` retrieves the hostname from the EC2 metadata, and `{local_hostname}` uses the hostname from the network configuration file.

  If you omit this field, the value of the `log_stream_name` parameter in the global `logs` section is used. If that is also omitted, the default value of `{instance_id}` is used.

  If a log stream doesn't already exist, it's created automatically.

- `retention_in_days` – Optional. Specifies the number of days to retain the log events in the specified log group.
  - If the agent is creating this log group now, and you omit this field, the retention of this new log group is set to never expire.

- If this log group already exists and you specify this field, the new retention that you specify is used. If you omit this field for a log group that already exists, the log group's retention is not changed.

  The CloudWatch agent wizard uses `-1` as the default value for this field when it is used to create the agent configuration file and you don't specify a value for log retention. This `-1` value set by the wizard specifies that the events in the log group will never expire. However, manually editing this value to `-1` has no effect.

Valid values are 1, 3, 5, 7, 14, 30, 60, 90, 120, 150, 180, 365, 400, 545, 731, 1827, 2192, 2557, 2922, 3288, and 3653.

If you configure the agent to write multiple log streams to the same log group, specifying the `retention_in_days` in one place will set the log retention for the entire log group. If you specify `retention_in_days` for the same log group in multiple places, the retention is set if all of those values are equal. However, if different `retention_in_days` values are specified for the same log group in multiple places, the log retention will not be set and the agent will stop, returning an error.

> **Note**
>
> The agent's IAM role or IAM user must have the `logs:PutRetentionPolicy` for it to be able to set retention policies. For more information, see Allowing the CloudWatch agent to set log retention policy.

> **Warning**
>
> If you set `retention_in_days` for a log group that already exists, all logs in that log group that were published before the number of days that you specify are deleted. For example, setting it to 3 would cause all logs from 3 days ago and before to be deleted.

- `filters` – Optional. Can contain an array of entries, each of which specifies a regular expression and a filter type to specify whether to publish or drop log entries that match the filter. If you omit this field, all logs in the log file are published to CloudWatch Logs. If you include this field, the agent processes each log message with all of the filters that you specify, and only the log events that pass all of the filters are published to CloudWatch

Logs. The log entries that don’t pass all of the filters will still remain in the host's log file, but will not be sent to CloudWatch Logs.

Each entry in the filters array can include the following fields:

- `type`– Denotes the type of filter. Valid values are `include` and `exclude`. With `include`, the log entry must match the expression to be published to CloudWatch Logs. With `exclude`, each log entry that matches the filter is not sent to CloudWatch Logs.
- `expression`– A regular expression string that follows the RE2 Syntax.

> **Note**
>
> The CloudWatch agent doesn't check the performance of any regular expression that you supply, or restrict the run time of the evaluation of the regular expressions. We recommend that you are careful not to write an expression that is expensive to evaluate. For more information about possible issues, see Regular expression Denial of Service - ReDoS

For example, the following excerpt of the CloudWatch agent configuration file publishes logs that are PUT and POST requests to CloudWatch Logs, but excluding logs that come from Firefox.

```
"collect_list": [
  {
    "file_path": "/opt/aws/amazon-cloudwatch-agent/logs/test.log",
    "log_group_name": "test.log",
    "log_stream_name": "test.log",
    "filters": [
      {
        "type": "exclude",
        "expression": "Firefox"
      },
      {
        "type": "include",
        "expression": "P(UT|OST)"
      }
    ]
  },
  .....
]
```

> **Note**
>
> The order of the filters in the configuration file matters for performance. In the preceding example, the agent drops all the logs that match `Firefox` before it starts evaluating the second filter. To cause fewer log entries to be evaluated by more than one filter, put the filter that you expect to rule out more logs first in the configuration file.

- `timezone` – Optional. Specifies the time zone to use when putting timestamps on log events. The valid values are `UTC` and `Local`. The default value is `Local`.

  This parameter is ignored if you don't specify a value for `timestamp_format`.

- `timestamp_format` – Optional. Specifies the timestamp format, using plaintext and special symbols that start with %. If you omit this field, the current time is used. If you use this field, you can use the symbols in the following list as part of the format.

  > **Note**
  >
  > This parameter is not considered when the `file_path` is set to `amazon-cloudwatch-agent.log`

  If a single log entry contains two time stamps that match the format, the first time stamp is used.

  This list of symbols is different than the list used by the older CloudWatch Logs agent. For a summary of these differences, see [Timestamp differences between the unified CloudWatch agent and the earlier CloudWatch Logs agent]

  `%y`

  Year without century as a zero-padded decimal number. For example, `19` to represent 2019.

  `%Y`

  Year with century as a decimal number. For example, `2019`.

`%b`

Month as the locale's abbreviated name

`%B`

Month as the locale's full name

`%m`

Month as a zero-padded decimal number

`%-m`

Month as a decimal number (not zero-padded)

`%d`

Day of the month as a zero-padded decimal number

`%-d`

Day of the month as a decimal number (not zero-padded)

`%A`

Full name of weekday, such as `Monday`

`%a`

Abbreviation of weekday, such as `Mon`

`%H`

Hour (in a 24-hour clock) as a zero-padded decimal number

`%I`

Hour (in a 12-hour clock) as a zero-padded decimal number

`%-I`

Hour (in a 12-hour clock) as a decimal number (not zero-padded)

`%p`

AM or PM

`%M`

Minutes as a zero-padded decimal number

`%-M`

Minutes as a decimal number (not zero-padded)

`%S`

Seconds as a zero-padded decimal number

`%-S`

Seconds as a decimal number (not zero padded)

`%f`

Fractional seconds as a decimal number (1-9 digits), zero-padded on the left.

`%Z`

Time zone, for example PST

`%z`

Time zone, expressed as the offset between the local time zone and UTC. For example, `-0700`. Only this format is supported. For example, `-07:00` isn't a valid format.

- `multi_line_start_pattern` – Specifies the pattern for identifying the start of a log message. A log message is made of a line that matches the pattern and any subsequent lines that don't match the pattern.

  If you omit this field, multi-line mode is disabled, and any line that begins with a non-whitespace character closes the previous log message and starts a new log message.

  If you include this field, you can specify `{timestamp_format}` to use the same regular expression as your timestamp format. Otherwise, you can specify a different regular expression for CloudWatch Logs to use to determine the start lines of multi-line entries.
- `encoding` – Specified the encoding of the log file so that it can be read correctly. If you specify an incorrect coding, there might be data loss because characters that can't be decoded are replaced with other characters.

  The default value is `utf-8`. The following are all possible values:

    ascii, big5, euc-jp, euc-kr, gbk, gb18030, ibm866, iso2022-jp, iso8859-2, iso8859-3, iso8859-4, iso8859-5, iso8859-6, iso8859-7, iso8859-8, iso8859-8-i, iso8859-10, iso8859-13, iso8859-14, iso8859-15, iso8859-16, koi8-r, koi8-u, macintosh, shift_jis, utf-8, utf-16, utf-16le, UTF-16, UTF-16LE, windows-874, windows-1250, windows-1251, windows-1252, windows-1253, windows-1254, windows-1255, windows-1256, windows-1257, windows-1258, x-mac-cyrillic

    - `service.name` – Optional. Specifies the service name to be used to populate the entity for finding related telemetry.
    - `deployment.environment` – Optional. Specifies the environment name to be used to populate the entity for finding related telemetry.

- The `windows_events` section specifies the type of Windows events to collect from servers running Windows Server. It includes the following fields:
  - `collect_list` – Required if `windows_events` is included. Specifies the types and levels of Windows events to be collected. Each log to be collected has an entry in this section, which can include the following fields:
    - `event_name` – Specifies the type of Windows events to log. This is equivalent to the Windows event log channel name: for example, `System`, `Security`, `Application`, and so on. This field is required for each type of Windows event to log.

      Note

      When CloudWatch retrieves messages from a Windows log channel, it looks up the log channel based on its `Full Name` property. Meanwhile, the Windows Event Viewer navigation pane displays the `Log Name` property of log channels. The `Full Name` and `Log Name` do not always match. To confirm the `Full Name` of a channel, right-click on it in the Windows Event viewer and open **Properties**.

    - `event_levels` – Specifies the levels of event to log. You must specify each level to log. Possible values include `INFORMATION`, `WARNING`, `ERROR`, `CRITICAL`, and `VERBOSE`. This field is required for each type of Windows event to log.
    - `log_group_name` – Required. Specifies what to use as the log group name in CloudWatch Logs.

- `log_stream_name` – Optional. Specifies what to use as the log stream name in CloudWatch Logs. As part of the name, you can use `{instance_id}`, `{hostname}`, `{local_hostname}`, and `{ip_address}` as variables within the name. `{hostname}` retrieves the hostname from the EC2 metadata, and `{local_hostname}` uses the hostname from the network configuration file.

  If you omit this field, the value of the `log_stream_name` parameter in the global `logs` section is used. If that is also omitted, the default value of `{instance_id}` is used.

  If a log stream doesn't already exist, it's created automatically.
- `event_format` – Optional. Specifies the format to use when storing Windows events in CloudWatch Logs. `xml` uses the XML format as in Windows Event Viewer. `text` uses the legacy CloudWatch Logs agent format.
- `retention_in_days` – Optional. Specifies the number of days to retain the Windows events in the specified log group.
  - If the agent is creating this log group now, and you omit this field, the retention of this new log group is set to never expire.
  - If this log group already exists and you specify this field, the new retention that you specify is used. If you omit this field for a log group that already exists, the log group's retention is not changed.

    The CloudWatch agent wizard uses `-1` as the default value for this field when it is used to create the agent configuration file and you don't specify a value for log retention. This `-1` value specifies set by the wizard specifies that the events in the log group don't expire. However, manually editing this value to `-1` has no effect.

  Valid values are 1, 3, 5, 7, 14, 30, 60, 90, 120, 150, 180, 365, 400, 545, 731, 1827, 2192, 2557, 2922, 3288, and 3653.

  If you configure the agent to write multiple log streams to the same log group, specifying the `retention_in_days` in one place will set the log retention for the entire log group. If you specify `retention_in_days` for the same log group in multiple places, the retention is set if all of those values are equal. However, if different `retention_in_days` values are specified for the same log group in multiple places, the log retention will not be set and the agent will stop, returning an error.

> **Note**
>
> The agent's IAM role or IAM user must have the `logs:PutRetentionPolicy` for it to be able to set retention policies. For more information, see [Allowing the CloudWatch agent to set log retention policy]().

> **Warning**
>
> If you set `retention_in_days` for a log group that already exists, all logs in that log group that were published before the number of days that you specify are deleted. For example, setting it to 3 would cause all logs from 3 days ago and before to be deleted.

- `log_stream_name` – Optional. Specifies the default log stream name to be used for any logs or Windows events that don't have individual log stream names defined in the `log_stream_name` parameter within their entry in `collect_list`.
- `endpoint_override` – Specifies a FIPS endpoint or private link to use as the endpoint where the agent sends logs. Specifying this field and setting a private link enables you to send the logs to an Amazon VPC endpoint. For more information, see [What Is Amazon VPC?]().

  The value of `endpoint_override` must be a string that is a URL.

  For example, the following part of the logs section of the configuration file sets the agent to use a VPC Endpoint when sending logs.

```
{
  "logs": {
    "endpoint_override": "vpce-XXXXXXXXXXXXXXXXXXXXXXXXX.logs.us-
east-1.vpce.amazonaws.com",
   ......
   },
}
```

- `force_flush_interval` – Specifies in seconds the maximum amount of time that logs remain in the memory buffer before being sent to the server. No matter the setting for this field, if the size of the logs in the buffer reaches 1 MB, the logs are immediately sent to the server. The default value is 5.

If you are using the agent to report high-resolution metrics in embedded metric format, and you are setting alarms on those metrics, keep this parameter set to the default value of 5. Otherwise, the metrics are reported with a delay that can cause alarming on partial or incomplete data.

- `credentials` – Specifies an IAM role to use when sending logs to a different Amazon account. If specified, this field contains one parameter, `role_arn`.
  - `role_arn` – Specifies the ARN of an IAM role to use for authentication when sending logs to a different Amazon account. For more information, see Sending metrics, logs, and traces to a different account. If specified here, this overrides the `role_arn` specified in the `agent` section of the configuration file, if any.
- `metrics_collected` – This field can contain sections to specify that the agent is to collect logs to enable use cases such as CloudWatch Application Signals and Container Insights with enhanced observability for Amazon EKS.
  - `application_signals` (Optional) Specifies that you want to enable CloudWatch Application Signals For more information about this configuration, see Enable CloudWatch Application Signals.
  - `kubernetes` – This field can contain an `enhanced_container_insights` parameter, which you can use to enable Container Insights with enhanced observability for Amazon EKS.
    - `enhanced_container_insights` – Set this to `true` to enable Container Insights with enhanced observability for Amazon EKS. For more information, see Container Insights with enhanced observability for Amazon EKS.
    - `accelerated_compute_metrics` – Set this to `false` to opt out of collecting Nvidia GPU metrics on Amazon EKS clusters. For more information, see NVIDIA GPU metrics.
  - `emf` – To collect metrics embedded in logs, it is no longer necessary to add this `emf` field. This is a legacy field that specified that the agent is to collect logs that are in embedded metric format. You can generate metric data from these logs. For more information, see Embedding metrics within logs.
  - `otlp` – Optional. Specifies that you want to collect metrics from the OpenTelemetry SDK. For more information about the fields that you can use in this section, see Collect metrics and traces with OpenTelemetry.

The following is an example of a `logs` section.

```
"logs":
   {
```

```
        "logs_collected": {
            "files": {
                "collect_list": [
                    {
                        "file_path": "c:\\ProgramData\\Amazon\\AmazonCloudWatchAgent\
\Logs\\amazon-cloudwatch-agent.log",
                        "log_group_name": "amazon-cloudwatch-agent.log",
                        "log_stream_name": "my_log_stream_name_1"
                    },
                    {
                        "file_path": "c:\\ProgramData\\Amazon\\AmazonCloudWatchAgent\
\Logs\\test.log",
                        "log_group_name": "test.log",
                        "log_stream_name": "my_log_stream_name_2"
                    }
                ]
            },
            "windows_events": {
                "collect_list": [
                    {
                        "event_name": "System",
                        "event_levels": [
                            "INFORMATION",
                            "ERROR"
                        ],
                        "log_group_name": "System",
                        "log_stream_name": "System"
                    },
                    {
                        "event_name": "CustomizedName",
                        "event_levels": [
                            "INFORMATION",
                            "ERROR"
                        ],
                        "log_group_name": "CustomizedLogGroup",
                        "log_stream_name": "CustomizedLogStream"
                    }
                ]
            }
        },
        "log_stream_name": "my_log_stream_name",
        "metrics_collected": {
            "kubernetes": {
                 "enhanced_container_insights": true
```

```
      }
    }
  }
```

CloudWatch agent configuration file: Traces section

By adding a `traces` section to the CloudWatch agent configuration file, you can enable CloudWatch Application Signals or collect traces from X-Ray and from the OpenTelemetry instrumentation SDK and send them to X-Ray.

> **Important**
>
> The agent's IAM role or IAM user must have the **AWSXrayWriteOnlyAccess** policy to send trace data to X-Ray. For more information, see [Create IAM roles and users for use with CloudWatch agent](#).

For a quick start for collecting traces, you can add just the following to the CloudWatch agent configuration file.

```
"traces_collected": {
        "xray": {
        },
        "otlp": {
        }
      }
```

If you add the previous section to the CloudWatch agent configuration file and restart the agent, this causes the agent to start collecting traces using the following default options and values. For more information about these parameters, see the parameter definitions later in this section.

```
"traces_collected": {
        "xray": {
            "bind_address": "127.0.0.1:2000",
            "tcp_proxy": {
              "bind_address": "127.0.0.1:2000"
            }
        },
        "otlp": {
            "grpc_endpoint": "127.0.0.1:4317",
            "http_endpoint": "127.0.0.1:4318"
```

```
        }
      }
```

The `traces` section can include the following fields:

- `traces_collected` – Required if the `traces` section is included. Specifies which SDKs to collect traces from. It can include the following fields:
  - `application_signals` – Optional. Specifies that you want to enable CloudWatch Application Signals For more information about this configuration, see Enable CloudWatch Application Signals.
  - `xray` – Optional. Specifies that you want to collect traces from the X-Ray SDK. This section can include the following fields:
    - `bind_address` – Optional. Specifies the UDP address for the CloudWatch agent to use to listen for X-Ray traces. The format is `ip:port`. This address must match the address set in the X-Ray SDK.

      If you omit this field, the default of `127.0.0.1:2000` is used.
    - `tcp_proxy` – Optional. Configures the address for a proxy used to support X-Ray remote sampling. For more information, see Configuring sampling rules in the X-Ray documentation.

      This section can contain the following field.
      - `bind_address` – Optional. Specifies the TCP address to which the CloudWatch agent should set up the proxy. The format is `ip:port`. This address must match the address set in the X-Ray SDK.

        If you omit this field, the default of `127.0.0.1:2000` is used.
  - `otlp` – Optional. Specifies that you want to collect traces from the OpenTelemetry SDK. For more information about the fields that you can use in this section, see Collect metrics and traces with OpenTelemetry). For more information about the Amazon Distro for OpenTelemetry, see Amazon Distro for OpenTelemetry. For more information about the Amazon Distro for OpenTelemetry SDKs, see Introduction.

    This section can include the following fields:
    - `grpc_endpoint` – Optional. Specifies the address for the CloudWatch agent to use to listen for OpenTelemetry traces sent using gRPC Remote Procedure Calls. The format is `ip:port`. This address must match the address set for the gRPC exporter in the OpenTelemetry SDK.

If you omit this field, the default of 127.0.0.1:4317 is used.

- http_endpoint – Optional. Specifies the address for the CloudWatch agent to use to listen for OTLP traces sent over HTTP. The format is ip:port. This address must match the address set for the HTTP exporter in the OpenTelemetry SDK.

  If you omit this field, the default of 127.0.0.1:4318 is used.

- concurrency – Optional. Specifies the maximum number of concurrent calls to X-Ray that can be used to upload traces. The default value is 8
- local_mode – Optional. If true, the agent doesn't collect Amazon EC2 instance metadata. The default is false
- endpoint_override – Optional. Specifies a FIPS endpoint or private link to use as the endpoint where the CloudWatch agent sends traces. Specifying this field and setting a private link enables you to send the traces to an Amazon VPC endpoint. For more information, see What is Amazon VPC

  The value of endpoint_override must be a string that is a URL.

- region_override – Optional. Specifies the Region to use for the X-Ray endpoint. The CloudWatch agent sends the traces to X-Ray in the specified Region. If you omit this field, the agent sends the traces to the Region where the Amazon EC2 instance is located.

  If you specify a Region here, it takes precedence over the setting of the region parameter in the agent section of the configuration file.

- proxy_override – Optional. Specifies the proxy server address for the CloudWatch agent to use when sending requests to X-Ray. The proxy server's protocol must be specified as part of this address.
- credentials – Specifies an IAM role to use when sending traces to a different Amazon account. If specified, this field contains one parameter, role_arn.
  - role_arn – Specifies the ARN of an IAM role to use for authentication when sending traces to a different Amazon account. For more information, see Sending metrics, logs, and traces to a different account. If specified here, this overrides the role_arn specified in the agent section of the configuration file, if any.
- transit_spans_in_otlp_format – Optional. If true, sends traces to X-Ray in the OpenTelemetry Protocol format, which supports span events in Transaction Search. For more information, see Adding custom attributes. The default is false.

CloudWatch agent configuration file: Complete examples

The following is an example of a complete CloudWatch agent configuration file for a Linux server.

The items listed in the `measurement` sections for the metrics you want to collect can either specify the complete metric name such or just the part of the metric name that will be appended to the type of resource. For example, specifying either `reads` or `diskio_reads` in the `measurement` section of the `diskio` section will cause the `diskio_reads` metric to be collected.

This example includes both ways of specifying metrics in the `measurement` section.

```
    {
      "agent": {
        "metrics_collection_interval": 10,
        "logfile": "/opt/aws/amazon-cloudwatch-agent/logs/amazon-cloudwatch-agent.log"
      },
      "metrics": {
        "namespace": "MyCustomNamespace",
        "metrics_collected": {
          "cpu": {
            "resources": [
              "*"
            ],
            "measurement": [
              {"name": "cpu_usage_idle", "rename": "CPU_USAGE_IDLE", "unit":
  "Percent"},
              {"name": "cpu_usage_nice", "unit": "Percent"},
              "cpu_usage_guest"
            ],
            "totalcpu": false,
            "metrics_collection_interval": 10,
            "append_dimensions": {
              "customized_dimension_key_1": "customized_dimension_value_1",
              "customized_dimension_key_2": "customized_dimension_value_2"
            }
          },
          "disk": {
            "resources": [
              "/",
              "/tmp"
            ],
            "measurement": [
              {"name": "free", "rename": "DISK_FREE", "unit": "Gigabytes"},
```

```
          "total",
          "used"
        ],
         "ignore_file_system_types": [
          "sysfs", "devtmpfs"
        ],
        "metrics_collection_interval": 60,
        "append_dimensions": {
          "customized_dimension_key_3": "customized_dimension_value_3",
          "customized_dimension_key_4": "customized_dimension_value_4"
        }
      },
      "diskio": {
        "resources": [
          "*"
        ],
        "measurement": [
          "reads",
          "writes",
          "read_time",
          "write_time",
          "io_time"
        ],
        "metrics_collection_interval": 60
      },
      "swap": {
        "measurement": [
          "swap_used",
          "swap_free",
          "swap_used_percent"
        ]
      },
      "mem": {
        "measurement": [
          "mem_used",
          "mem_cached",
          "mem_total"
        ],
        "metrics_collection_interval": 1
      },
      "net": {
        "resources": [
          "eth0"
        ],
```

```
          "measurement": [
            "bytes_sent",
            "bytes_recv",
            "drop_in",
            "drop_out"
          ]
        },
        "netstat": {
          "measurement": [
            "tcp_established",
            "tcp_syn_sent",
            "tcp_close"
          ],
          "metrics_collection_interval": 60
        },
        "processes": {
          "measurement": [
            "running",
            "sleeping",
            "dead"
          ]
        }
      },
      "append_dimensions": {
        "ImageId": "${aws:ImageId}",
        "InstanceId": "${aws:InstanceId}",
        "InstanceType": "${aws:InstanceType}",
        "AutoScalingGroupName": "${aws:AutoScalingGroupName}"
      },
      "aggregation_dimensions" : [["ImageId"], ["InstanceId", "InstanceType"],
 ["d1"],[]],
      "force_flush_interval" : 30
    },
    "logs": {
      "logs_collected": {
        "files": {
          "collect_list": [
            {
              "file_path": "/opt/aws/amazon-cloudwatch-agent/logs/amazon-cloudwatch-
agent.log",
              "log_group_name": "amazon-cloudwatch-agent.log",
              "log_stream_name": "amazon-cloudwatch-agent.log",
              "timezone": "UTC"
            },
```

```
              {
                "file_path": "/opt/aws/amazon-cloudwatch-agent/logs/test.log",
                "log_group_name": "test.log",
                "log_stream_name": "test.log",
                "timezone": "Local"
              }
            ]
          }
        },
        "log_stream_name": "my_log_stream_name",
        "force_flush_interval" : 15,
        "metrics_collected": {
           "kubernetes": {
                "enhanced_container_insights": true
      }
    }
  }
}
```

The following is an example of a complete CloudWatch agent configuration file for a server running Windows Server.

```
{
      "agent": {
        "metrics_collection_interval": 60,
        "logfile": "c:\\ProgramData\\Amazon\\AmazonCloudWatchAgent\\Logs\\amazon-
cloudwatch-agent.log"
      },
      "metrics": {
        "namespace": "MyCustomNamespace",
        "metrics_collected": {
          "Processor": {
            "measurement": [
              {"name": "% Idle Time", "rename": "CPU_IDLE", "unit": "Percent"},
              "% Interrupt Time",
              "% User Time",
              "% Processor Time"
            ],
            "resources": [
              "*"
            ],
            "append_dimensions": {
              "customized_dimension_key_1": "customized_dimension_value_1",
```

```
            "customized_dimension_key_2": "customized_dimension_value_2"
          }
        },
        "LogicalDisk": {
          "measurement": [
            {"name": "% Idle Time", "unit": "Percent"},
            {"name": "% Disk Read Time", "rename": "DISK_READ"},
            "% Disk Write Time"
          ],
          "resources": [
            "*"
          ]
        },
        "customizedObjectName": {
          "metrics_collection_interval": 60,
          "customizedCounterName": [
            "metric1",
            "metric2"
          ],
          "resources": [
            "customizedInstances"
          ]
        },
        "Memory": {
          "metrics_collection_interval": 5,
          "measurement": [
            "Available Bytes",
            "Cache Faults/sec",
            "Page Faults/sec",
            "Pages/sec"
          ]
        },
        "Network Interface": {
          "metrics_collection_interval": 5,
          "measurement": [
            "Bytes Received/sec",
            "Bytes Sent/sec",
            "Packets Received/sec",
            "Packets Sent/sec"
          ],
          "resources": [
            "*"
          ],
          "append_dimensions": {
```

```
              "customized_dimension_key_3": "customized_dimension_value_3"
            }
          },
          "System": {
            "measurement": [
              "Context Switches/sec",
              "System Calls/sec",
              "Processor Queue Length"
            ]
          }
        },
        "append_dimensions": {
          "ImageId": "${aws:ImageId}",
          "InstanceId": "${aws:InstanceId}",
          "InstanceType": "${aws:InstanceType}",
          "AutoScalingGroupName": "${aws:AutoScalingGroupName}"
        },
        "aggregation_dimensions" : [["ImageId"], ["InstanceId", "InstanceType"],
 ["d1"],[]]
      },
      "logs": {
        "logs_collected": {
          "files": {
            "collect_list": [
              {
                "file_path": "c:\\ProgramData\\Amazon\\AmazonCloudWatchAgent\\Logs\
\amazon-cloudwatch-agent.log",
                "log_group_name": "amazon-cloudwatch-agent.log",
                "timezone": "UTC"
              },
              {
                "file_path": "c:\\ProgramData\\Amazon\\AmazonCloudWatchAgent\\Logs\
\test.log",
                "log_group_name": "test.log",
                "timezone": "Local"
              }
            ]
          },
          "windows_events": {
            "collect_list": [
              {
                "event_name": "System",
                "event_levels": [
                  "INFORMATION",
```

```
                "ERROR"
              ],
              "log_group_name": "System",
              "log_stream_name": "System",
              "event_format": "xml"
            },
            {
              "event_name": "CustomizedName",
              "event_levels": [
                "WARNING",
                "ERROR"
              ],
              "log_group_name": "CustomizedLogGroup",
              "log_stream_name": "CustomizedLogStream",
              "event_format": "xml"
            }
          ]
        }
      },
      "log_stream_name": "example_log_stream_name"
    }
  }
```

Save the CloudWatch agent configuration file manually

If you create or edit the CloudWatch agent configuration file manually, you can give it any name. After you have created the file, you can copy it to other servers where you want to run the agent.

Uploading the CloudWatch agent configuration file to Systems Manager Parameter Store

If you plan to use the SSM Agent to install the CloudWatch agent on servers, after you manually edit the CloudWatch agent configuration file, you can upload it to Systems Manager Parameter Store. To do so, use the Systems Manager `put-parameter` command.

To be able to store the file in Parameter Store, you must use an IAM role with sufficient permissions. For more information, see [Create IAM roles and users for use with the CloudWatch agent](#).

Use the following command, where *`parameter name`* is the name to be used for this file in Parameter Store and *`configuration_file_pathname`* is the path and file name of the configuration file that you edited.

```
aws ssm put-parameter --name "parameter name" --type "String" --value
 file://configuration_file_pathname
```

Enable CloudWatch Application Signals

Use CloudWatch Application Signals to automatically instrument your applications on Amazon so that you can track application performance against your business objectives. Application Signals provides you a unified, application-centric view of your Java applications, their dependencies, and their edges. For more information, see Application Signals.

CloudWatch Application Signals leverages the CloudWatch agent to receive metrics and traces from your auto-instrumented applications, optionally apply rules to reduce high cardinality, and then publish the processed telemetry to CloudWatch. You can provide custom configuration to the CloudWatch agent specifically for Application Signals using the agent configuration file. To start with, the presence of an `application_signals` section under the `metrics_collected` section within the `logs` section of the agent configuration file specifies that the CloudWatch agent will receive metrics from your auto-instrumented applications. Similarly, the presence of an `application_signals` section under the `traces_collected` section within the `traces` section of the agent configuration file specifies that the CloudWatch agent is enabled to receive traces from your auto-instrumented applications. In addition, you can optionally pass in custom configuration rules to reduce publishing high-cardinality telemetry as outlined in this section.

- For Amazon EKS clusters, when you install the Amazon CloudWatch Observability EKS add-on, the CloudWatch agent is by default enabled to receive both metrics and traces from your auto-instrumented applications. If you would like to optionally pass in custom configuration rules, you can do so by passing in a custom agent configuration to the Amazon EKS add-on when you create or update it by using additional configuration, as outlined in (Optional) Additional configuration.
- For RedHat for OpenShift on Amazon (ROSA), when you install the CloudWatch agent operator using helm charts, the CloudWatch agent is by default enabled to receive both metrics and traces from your auto-instrumented applications. If you would like to optionally pass in custom configuration rules, you can do so by passing in a custom agent configuration by using the Helm chart, as outlined in (Optional) [Additional configuration], as outlined in (Optional) Additional configuration.
- For other supported platforms including Amazon EC2, you must start the CloudWatch agent with an agent configuration that enables Application Signals by specifying the

`application_signals` sections and optionally any custom configuration rules as outlined later in this section.

The following is an overview of the fields in the CloudWatch agent configuration file that are related to CloudWatch Application Signals.

- `logs`
  - `metrics_collected` – This field can contain sections to specify that the agent is to collect logs to enable use cases such as CloudWatch Application Signals and Container Insights with enhanced observability for Amazon EKS.

    > **Note**
    >
    > Previously this section was also used to specify that the agent is to collect logs that are in embedded metric format. Those settings are no longer needed.

    - `application_signals` (Optional) Specifies that you want to enable CloudWatch Application Signals to receive metrics from your auto-instrumented applications to facilitate CloudWatch Application Signals.
      - `rules` (Optional) An array of rules to conditionally select metrics and traces and apply actions to handle high-cardinality scenarios. Each rule can contain the following fields:
        - `rule_name` (Optional) The name of the rule.
        - `selectors` (Optional) An array of metrics and traces dimension matchers. Each selector must provide the following fields:
          - `dimension` Required if `selectors` is not empty. This specifies the dimension of metrics and traces to use as a filter.
          - `match` Required if `selectors` is not empty. A wildcard pattern used for matching values of the specified dimension.
        - `action` (Optional) The action to be applied to metrics and traces that match the specified selectors. The value of `action` must be one of the following keywords:
          - `keep` Specifies to send only the metrics and traces to CloudWatch if matched by the `selectors`.
          - `drop` Specifies to drop the metric and traces that match the `selectors`.

    - `replace` Specifies to replace the dimensions of the metrics and traces that match `selectors`. They are replaced according to the `replacements` section.
  - `replacements` Required if `action` is `replace`. An array of dimension and value pairs that will be applied to metrics and traces that match the specified `selectors` when the `action` is `replace`. Each replacement must provide the following fields:
    - `target_dimension` Required if `replacements` is not empty. Specifies the dimension that needs to be replace.
    - `value` Required if `replacements` is not empty. The value to replace the original value of `target_dimension` with.
- `limiter` (Optional) Use this section to limit how many metrics and dimensions Application Signals sends to CloudWatch, to optimize your costs.
  - `disabled` (Optional) If `true`, the metric limiting feature is disabled. The default is `false`
  - `drop_threshold` (Optional) The maximum number of distinct metrics per service in one rotation interval that can be exported by one CloudWatch agent. The default is 500.
  - `rotation_interval` (Optional) The interval at which the limiter resets the metric records for distinction counting. This is expressed as a string with a sequence of numbers and a unit suffix. Fractions are supported. The supported unit suffixes are `s`, `m`, `h`, `ms`, `us`, and `ns`

    The default is `1h` for one hour.
  - `log_dropped_metrics` (Optional) Specifies whether the agent should write logs to the CloudWatch agent logs when Application Signals metrics are dropped. The default is `false`.

    > **Note**
    >
    > To activate this logging, the `debug` parameter in the `agent` section must also be set to `true`.

- `traces`
  - `traces_collected`
    - `application_signals` Optional. Specify this to enable the CloudWatch agent to receive traces from your auto-instrumented applications for facilitating CloudWatch Application Signals.

> **Note**
>
> Even though the custom `application_signals` rules are specified under the `metrics_collected` section that is contained in the `logs` section, they also implicitly apply to the `traces_collected` section as well. The same set of rules will apply to both metrics and traces.

When there are multiple rules with different actions, they apply in the following sequence: `keep`, then `drop`, then `replace`.

The following is an example of a full CloudWatch agent configuration file that applies custom rules.

```
{
  "logs": {
    "metrics_collected": {
      "application_signals": {
        "rules": [
          {
            "rule_name": "keep01",
            "selectors": [
              {
                "dimension": "Service",
                "match": "pet-clinic-frontend"
              },
              {
                "dimension": "RemoteService",
                "match": "customers-service"
              }
            ],
            "action": "keep"
          },
          {
            "rule_name": "drop01",
            "selectors": [
              {
                "dimension": "Operation",
                "match": "GET /api/customer/owners/*"
              }
            ],
            "action": "drop"
```

```
          },
          {
            "rule_name": "replace01",
            "selectors": [
              {
                "dimension": "Operation",
                "match": "PUT /api/customer/owners/*/pets/*"
              },
              {
                "dimension": "RemoteOperation",
                "match": "PUT /owners"
              }
            ],
            "replacements": [
              {
                "target_dimension": "Operation",
                "value": "PUT /api/customer/owners/{ownerId}/pets{petId}"
              }
            ],
            "action": "replace"
          }
        ]
      }
    }
  },
  "traces": {
    "traces_collected": {
      "application_signals": {}
    }
  }
}
```

For the previous example configuration file, the `rules` are processed as follows:

1. Rule `keep01` ensures that any metrics and traces with the dimension `Service` as `pet-clinic-frontend` and the dimension `RemoteService` as `customers-service` are kept.
2. For the processed metrics and traces after applying `keep01`, the `drop01` rule ensures that metrics and traces with the dimension `Operation` as `GET /api/customer/owners/*` are dropped.
3. For the processed metrics and traces after applying `drop01`, the `replace01` rule updates metrics and traces that have the dimension `Operation` as `PUT /api/customer/owners/*/`

pets/* and the dimension RemoteOperation as PUT /owners such that their Operation dimension is now replaced to be PUT /api/customer/owners/{ownerId}/pets{petId}.

The following is a complete example of a CloudWatch configuration file that manages cardinality in Application Signals by changing the metric limit to 100, enabling the logging of dropped metrics, and setting the rotation interval to two hours.

```
{
    "logs": {
        "metrics_collected": {
            "application_signals": {
                "limiter": {
                    "disabled": false,
                    "drop_threshold": 100,
                    "rotation_interval": "2h",
                    "log_dropped_metrics": true
                }
            }
        },
        "traces": {
            "traces_collected": {
                "application_signals": {}
            }
        }
    }
}
```

Collect network performance metrics

EC2 instances running on Linux that use the Elastic Network Adapter (ENA) publish network performance metrics. Version 1.246396.0 and later of the CloudWatch agent enable you to import these network performance metrics into CloudWatch. When you import these network performance metrics into CloudWatch, they are charged as CloudWatch custom metrics.

For more information about the ENA driver, see Enabling enhanced networking with the Elastic Network Adapter (ENA) on Linux instances and Enabling enhanced networking with the Elastic Network Adapter (ENA) on Windows instances.

How you set up the collection of network performance metrics differs on Linux servers and Windows servers.

The following table lists these network performance metrics enabled by the ENA adapter. When the CloudWatch agent imports these metrics into CloudWatch from Linux instances, it prepends `ethtool_` at the beginning of each of these metric names.

Metric	Description
Name on Linux servers: **`bw_in_allowance_exceeded`** Name on Windows servers: **`Aggregate inbound BW allowance exceeded`**	The number of packets queued and/or dropped because the inbound aggregate bandwidth exceeded the maximum for the instance. This metric is collected only if you have listed it in the `ethtool` subsection of the `metrics_collected` section of the CloudWatch agent configuration file. For more information, see Collect network performance metrics Unit: None
Name on Linux servers: **`bw_out_allowance_exceeded`** Name on Windows servers: **`Aggregate outbound BW allowance exceeded`**	The number of packets queued and/or dropped because the outbound aggregate bandwidth exceeded the maximum for the instance. This metric is collected only if you have listed it in the `ethtool` subsection of the `metrics_collected` section of the CloudWatch agent configuration file. For more information, see Collect network performance metrics Unit: None
Name on Linux servers:**`conntrack_allowance_available`** Name on Windows servers: **`Available connection tracking allowance`**	Reports the number of tracked connections that can be established by the instance before hitting the Connections Tracked allowance of that instance type. This metric is available only on Nitro-based EC2 instances using the Linux driver for Elastic Network Adapter (ENA) starting from version 2.8.1, and on computers using the Windows driver for Elastic Network Adapter (ENA) starting from version 2.6.0.

Metric	Description
	This metric is collected only if you have listed it in the `ethtool` subsection of the `metrics_collected` section of the CloudWatch agent configuration file. For more information, see [Collect network performance metrics](#) Unit: None
Name on Linux servers:**`ena_srd_mode`** Name on Windows servers: **`ena srd mode`**	Describes which ENA Express features are enabled. For more information about ENA Express, see [Improve network performance with ENA Express on Linux instances](#) Values are as follows: • `0` = ENA Express off, UDP off • 1 = ENA Express on, UDP off • 2 = ENA Express off, UDP on **Note** This happens only when ENA Express was originally enabled, and UDP was configured to use it. The prior value is retained for UDP traffic. • 3 = ENA Express on, UDP on

Metric	Description
Name on Linux servers:**ena_srd_eligible_tx_pkts** Name on Windows servers: **ena srd eligible tx pkts**	The number of network packets sent within a given time period that meet Amazon Scalable Reliable Datagram (SRD) requirements for eligibility, as follows: • Both sending and receiving instance types are supported. • Both sending and receiving instances must have ENA Express configured. • The sending and receiving instances must be on the same subnet. • The network path between the instances must not include middleware boxes. ENA Express doesn't currently support middleware boxes.
Name on Linux servers:**ena_srd_tx_pkts** Name on Windows servers: **ena srd tx pkts**	The number of SRD packets transmitted within a given time period.
Name on Linux servers:**ena_srd_rx_pkts** Name on Windows servers: **ena srd rx pkts**	The number of SRD packets received within a given time period.
Name on Linux servers:**ena_srd_resource_utilization** Name on Windows servers: **ena srd resource utilization**	The percentage of the maximum allowed memory utilization for concurrent SRD connections that the instance has consumed.

Metric	Description
Name on Linux servers: **linklocal_allowance_exceeded** Name on Windows servers: **Link local packet rate allowance exceeded**	The number of packets dropped because the PPS of the traffic to local proxy services exceeded the maximum for the network interface. This impacts traffic to the DNS service, the Instance Metadata Service, and the Amazon Time Sync Service, but does not impact traffic to custom DNS resolvers. This metric is collected only if you have listed it in the `ethtool` subsection of the `metrics_collected` section of the CloudWatch agent configuration file. For more information, see [Collect network performance metrics] Unit: None
Name on Linux servers: **pps_allowance_exceeded** Name on Windows servers: **PPS allowance exceeded**	The number of packets queued and/or dropped because the bidirectional PPS exceeded the maximum for the instance. This metric is collected only if you have listed it in the `ethtool` subsection of the `metrics_collected` section of the CloudWatch agent configuration file. For more information, see [Collect network performance metrics] Unit: None

Linux setup

On Linux servers, the *ethtool plugin* enables you to import the network performance metrics into CloudWatch.

ethtool is a standard Linux utility that can collect statistics about Ethernet devices on Linux servers. The statistics it collects depend on the network device and driver. Examples of these statistics include `tx_cnt`, `rx_bytes`, `tx_errors`, and `align_errors`. When you use the ethtool plugin with the CloudWatch agent, you can also import these statistics into CloudWatch, along with the EC2 network performance metrics listed earlier in this section.

> **Tip**
>
> To find the statistics available on our operating system and network device, use the `ethtool -S` command.

When the CloudWatch agent imports metrics into CloudWatch, it adds an `ethtool_` prefix to the names of all imported metrics. So the standard ethtool statistic `rx_bytes` is called `ethtool_rx_bytes` in CloudWatch, and the EC2 network performance metric `bw_in_allowance_exceeded` is called `ethtool_bw_in_allowance_exceeded` in CloudWatch.

On Linux servers, to import ethtool metrics, add an `ethtool` section to the `metrics_collected` section of the CloudWatch agent configuration file. The `ethtool` section can include the following subsections:

- **interface_include**— Including this section causes the agent to collect metrics from only the interfaces that have names listed in this section. If you omit this section, metrics are collected from all Ethernet interfaces that aren't listed in `interface_exclude`.

  The default ethernet interface is `eth0`.
- **interface_exclude**— If you include this section, list the Ethernet interfaces that you don't want to collect metrics from.

  The ethtool plugin always ignores loopback interfaces.
- **metrics_include**— This section lists the metrics to import into CloudWatch. It can include both standard statistics collected by ethtool and Amazon EC2 high-resolution network metrics.

The following example displays part of the CloudWatch agent configuration file. This configuration collects the standard ethtool metrics `rx_packets` and `tx_packets`, and the Amazon EC2 network performance metrics from only the `eth1` interface.

For more information about the CloudWatch agent configuration file, see Manually create or edit the CloudWatch agent configuration file.

```
"metrics": {
```

```
    "append_dimensions": {
      "InstanceId": "${aws:InstanceId}"
    },
    "metrics_collected": {
      "ethtool": {
        "interface_include": [
          "eth1"
        ],
        "metrics_include": [
          "bw_in_allowance_exceeded",
          "bw_out_allowance_exceeded",
          "conntrack_allowance_exceeded",
          "linklocal_allowance_exceeded",
          "pps_allowance_exceeded"
         ]
      }
   }
}
```

Windows setup

On Windows servers, the network performance metrics are available through Windows Performance Counters, which the CloudWatch agent already collects metrics from. So you do not need a plugin to collect these metrics from Windows servers.

The following is a sample configuration file to collect network performance metrics from Windows. For more information about editing the CloudWatch agent configuration file, see Manually create or edit the CloudWatch agent configuration file.

```
{
    "metrics": {
        "append_dimensions": {
            "InstanceId": "${aws:InstanceId}"
        },
        "metrics_collected": {
            "ENA Packets Shaping": {
                "measurement": [
                    "Aggregate inbound BW allowance exceeded",
                    "Aggregate outbound BW allowance exceeded",
                    "Connection tracking allowance exceeded",
                    "Link local packet rate allowance exceeded",
                    "PPS allowance exceeded"
                ],
```

```
                "metrics_collection_interval": 60,
                "resources": [
                    "*"
                ]
            }
        }
    }
}
```

Viewing network performance metrics

After importing network performance metrics into CloudWatch, you can view these metrics as time series graphs, and create alarms that can watch these metrics and notify you if they breach a threshold that you specify. The following procedure shows how to view ethtool metrics as a time series graph. For more information about setting alarms, see [Using Amazon CloudWatch alarms](#).

Because all of these metrics are aggregate counters, you can use CloudWatch metric math functions such as `RATE(METRICS())` to calculate the rate for these metrics in graphs or use them to set alarms. For more information about metric math functions, see [Using math expressions with CloudWatch metrics](#).

To view network performance metrics in the CloudWatch console

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Metrics**.
3. Choose the namespace for the metrics collected by the agent. By default, this is **CWAgent**, but you may have specified a different namespace in the CloudWatch agent configuration file.
4. Choose a metric dimension (for example, **Per-Instance Metrics**).
5. The **All metrics** tab displays all metrics for that dimension in the namespace. You can do the following:

    a. To graph a metric, select the check box next to the metric. To select all metrics, select the check box in the heading row of the table.

    b. To sort the table, use the column heading.

    c. To filter by resource, choose the resource ID, and then choose **Add to search**.

    d. To filter by metric, choose the metric name, and then choose **Add to search**.

6. (Optional) To add this graph to a CloudWatch dashboard, choose **Actions**, and then choose **Add to dashboard**.

Collect NVIDIA GPU metrics

You can use the CloudWatch agent to collect NVIDIA GPU metrics from Linux servers. To set this up, add a `nvidia_gpu` section inside the `metrics_collected` section of the CloudWatch agent configuration file. For more information, see Linux section.

Additionally, the instance must have an NVIDIA driver installed. NVIDIA drivers on pre-installed on some Amazon Machine Images (AMIs). Otherwise, you can manually install the driver. For more information, see Install NVIDIA drivers on Linux instances.

The following metrics can be collected. All of these metrics are collected with no CloudWatch `Unit`, but you can specify a unit for each metric by adding a parameter to the CloudWatch agent configuration file. For more information, see Linux section.

Metric	Metric name in CloudWatch	Description
`utilization_gpu`	`nvidia_smi_utilization_gpu`	The percentage of time over the past sample period during which one or more kernals on the GPU was running.
`temperature_gpu`	`nvidia_smi_temperature_gpu`	The core GPU temperature in degrees Celsius.
`power_draw`	`nvidia_smi_power_draw`	The last measured power draw for the entire board, in watts.
`utilization_memory`	`nvidia_smi_utilization_memory`	The percentage of time over the past sample period during which global (device) memory was being read or written.
`fan_speed`	`nvidia_smi_fan_speed`	The percentage of maximum fan speed that the device's fan is currently intended to run at.
`memory_total`	`nvidia_smi_memory_total`	Reported total memory, in MB.
`memory_used`	`nvidia_smi_memory_used`	Memory used, in MB.

Metric	Metric name in CloudWatch	Description
memory_free	nvidia_smi_memory_free	Memory free, in MB.
pcie_link_gen_current	nvidia_smi_pcie_link_gen_current	The current link generation.
pcie_link_width_current	nvidia_smi_pcie_link_width_current	The current link width.
encoder_stats_session_count	nvidia_smi_encoder_stats_session_count	Current number of encoder sessions.
encoder_stats_average_fps	nvidia_smi_encoder_stats_average_fps	The moving average of the encode frames per second.
encoder_stats_average_latency	nvidia_smi_encoder_stats_average_latency	The moving average of the encode latency in microseconds.
clocks_current_graphics	nvidia_smi_clocks_current_graphics	The current frequency of the graphics (shader) clock.
clocks_current_sm	nvidia_smi_clocks_current_sm	The current frequency of the Streaming Multiprocessor (SM) clock.
clocks_current_memory	nvidia_smi_clocks_current_memory	The current frequency of the memory clock.

Metric	Metric name in CloudWatch	Description
clocks_current_video	nvidia_smi_clocks_current_video	The current frequency of the video (encoder plus decoder) clocks.

All of these metrics are collected with the following dimensions:

Dimension	Description
index	A unique identifier for the GPU on this server. Represents the NVIDIA Management Library (NVML) index of the device.
name	The type of GPU. For example, `NVIDIA Tesla A100`
arch	The server architecture.

Collect Java Management Extensions (JMX) metrics

You can use the CloudWatch agent to collect Java Management Extensions (JMX) metrics from your Java applications.

The CloudWatch agent supports collecting these metrics from the following versions:

- JVM 8 and later
- Kafka 0.8.2.x and later
- Tomcat 9, 10.1, and 11 (beta)

Amazon EC2

To enable JMX in your JVM instance

For the CloudWatch agent to be able to collect JMX metrics, your application's JVM must bind to a port using the `com.sun.management.jmxremote.port` system property.

```
java -Dcom.sun.management.jmxremote.port=port-number -jar example.jar
```

For more information and other configurations, see the JMX documentation.

Amazon EKS

To enable JMX on your Java application pods

When using the CloudWatch Observability EKS add-on, you can manage how JMX metrics are enabled with annotations. For more information, see Install the CloudWatch agent with the Amazon CloudWatch Observability EKS add-on or the Helm chart. To enable JMX metrics collection from a workload, add the following annotations to the workload manifest file under the `PodTemplate` section:

- `instrumentation.opentelemetry.io/inject-java: "true"`
- One or more of the following:
  - For JVM metrics: `cloudwatch.aws.amazon.com/inject-jmx-jvm: "true"`
  - For Kafka broker metrics: `cloudwatch.aws.amazon.com/inject-jmx-kafka: "true"`
  - For Kafka consumer metrics: `cloudwatch.aws.amazon.com/inject-jmx-kafka-consumer: "true"`
  - For Kafka producer metrics: `cloudwatch.aws.amazon.com/inject-jmx-kafka-producer: "true"`
  - For Tomcat metrics: `cloudwatch.aws.amazon.com/inject-jmx-tomcat: "true"`

To start collecting JMX metrics, add a `jmx` section inside the `metrics_collected` section of the CloudWatch agent configuration file. The `jmx` section can contain the following fields.

- `jvm` – Optional. Specifies that you want to retrieve Java Virtual Machine (JVM) metrics from the instance. For more information, see Collect JVM metrics.

  This section can include the following fields:

  - `measurement` – Specifies the array of JVM metrics to be collected. For a list of the possible values to use here, see the **Metric** column in the table in Collect JVM metrics.

    Within the entry for each individual metric, you can optionally specify one or both of the following:

    - `rename` – Specifies a different name for this metric.
    - `unit` – Specifies the unit to use for this metric, overriding the default unit for the metric. The unit that you specify must be a valid CloudWatch metric unit, as listed in the `Unit` description in MetricDatum.

- `kafka` – Optional. Specifies that you want to retrieve Apache Kafka broker metrics from the instance. For more information, see Collect Kafka metrics.

  This section can include the following fields:

  - `measurement` – Specifies the array of Kafka broker metrics to be collected. For a list of the possible values to use here, see the **Metric** column in the first table in Collect Kafka metrics.

    Within the entry for each individual metric, you can optionally specify one or both of the following:

    - `rename` – Specifies a different name for this metric.
    - `unit` – Specifies the unit to use for this metric, overriding the default unit for the metric. The unit that you specify must be a valid CloudWatch metric unit, as listed in the `Unit` description in MetricDatum.

- `kafka-consumer` – Optional. Specifies that you want to retrieve Apache Kafka consumer metrics from the instance. For more information, see Collect Kafka metrics.

  This section can include the following fields:

  - `measurement` – Specifies the array of Kafka broker metrics to be collected. For a list of the possible values to use here, see the **Metric** column in the second metrics table in Collect Kafka metrics.

Within the entry for each individual metric, you can optionally specify one or both of the following:

- `rename` – Specifies a different name for this metric.
- `unit` – Specifies the unit to use for this metric, overriding the default unit for the metric. The unit that you specify must be a valid CloudWatch metric unit, as listed in the `Unit` description in MetricDatum.

- `kafka-producer` – Optional. Specifies that you want to retrieve Apache Kafka producer metrics from the instance. For more information, see Collect Kafka metrics.

  This section can include the following fields:

  - `measurement` – Specifies the array of Kafka broker metrics to be collected. For a list of the possible values to use here, see the **Metric** column in the third metrics table in Collect Kafka metrics.

    Within the entry for each individual metric, you can optionally specify one or both of the following:

    - `rename` – Specifies a different name for this metric.
    - `unit` – Specifies the unit to use for this metric, overriding the default unit for the metric. The unit that you specify must be a valid CloudWatch metric unit, as listed in the `Unit` description in MetricDatum.

- `tomcat` – Optional. Specifies that you want to retrieve Tomcat metrics from the instance. For more information, see Collect Tomcat metrics.

  This section can include the following fields:

  - `measurement` – Specifies the array of Tomcat metrics to be collected. For a list of the possible values to use here, see the **Metric** column in the table in Collect Tomcat metrics.

    Within the entry for each individual metric, you can optionally specify one or both of the following:

    - `rename` – Specifies a different name for this metric.
    - `unit` – Specifies the unit to use for this metric, overriding the default unit for the metric. The unit that you specify must be a valid CloudWatch metric unit, as listed in the `Unit` description in MetricDatum.

The `jmx` section can also include the optional `append_dimensions` field:

- `append_dimensions` – Optional. Additional dimensions to use for only the process metrics. If you specify this field, it's used in addition to dimensions specified in the `append_dimensions` field that is used for all types of metrics collected by the agent.

The following fields are for Amazon EC2 only.

- `endpoint` – The address for the JMX client to connect to. The format is `ip:port`. If the endpoint is not the localhost, and password authentication and SSL must be enabled.
- `metrics_collection_interval` – Optional. Specifies how often to collect the processes metrics, overriding the global `metrics_collection_interval` specified in the `agent` section of the configuration file.

  This value is specified in seconds. For example, specifying 10 causes metrics to be collected every 10 seconds, and setting it to 300 specifies metrics to be collected every 5 minutes.

  If you set this value below 60 seconds, each metric is collected as a high-resolution metric. For more information, see [High-resolution metrics](#).

If JMX was enabled with password authentication or SSL for remote access, you can use the following fields.

- `password_file` – Optional. Specifies a Java properties file of keys to passwords. The file must be read-only and restricted to the user running the CloudWatch agent. If password authentication is enabled, this requires the same username and password pair as the entry in the JMX password file provided in the `com.sun.management.jmxremote.password.file` property. If SSL is enabled, it requires entries for `keystore` and `truststore` and corresponds to the `javax.net.ssl.keyStorePassword` and `javax.net.ssl.trustStorePassword` respectively.
- `username` – If password authentication is enabled, specify the username that matches the username in the provided password file.
- `keystore_path` – If SSL is enabled, specify the full path to the Java keystore, which consists of a private key and a certificate to the public key. Corresponds to the `javax.net.ssl.keyStore` property.
- `keystore_type` – If SSL is enabled, specify the type of keystore being used. Corresponds to the `javax.net.ssl.keyStoreType` property.

- `truststore_path` – If SSL is enabled, specify the full path to the Java truststore, which must contain the remote JMX server's public certificate. Corresponds to the `javax.net.ssl.trustStore` property.
- `truststore_type` – If SSL is enabled, specify the type of truststore being used. Corresponds to the `javax.net.ssl.trustStoreType` property.
- `remote_profile` – Optional. Supported JMX remote profiles are TLS in combination with SASL profiles: `SASL/PLAIN`, `SASL/DIGEST-MD5`, and `SASL/CRAM-MD5`. Should be one of:`SASL/PLAIN`, `SASL/DIGEST-MD5`, `SASL/CRAM-MD5`, `TLS SASL/PLAIN`, `TLS SASL/DIGEST-MD5`, or `TLS SASL/CRAM-MD5`
- `realm` – Optional. The realm as required by the remote profile `SASL/DIGEST-MD5`.
- `registry_ssl_enabled` – If RMI registry authentication is enabled. Set to true if the JVM was configured with `com.sun.management.jmxremote.registry.ssl=true`.
- `insecure` Set to `true` to opt out of the validation required if the agent is configured for a non-localhost endpoint.

The following is an example of the `jmx` section of the CloudWatch agent configuration file.

```
{
  "metrics": {
    "metrics_collected": {
      "jmx": [
        {
          "endpoint": "remotehost:1314",
          "jvm": {
            "measurement": [
              "jvm.memory.heap.init",
              "jvm.memory.nonheap.used"
            ]
          },
          "kafka": {
            "measurement": [
              "kafka.request.count",
              {
                "name": "kafka.message.count",
                "rename": "KAFKA_MESSAGE_COUNT",
                "unit": "Count"
              }
            ]
          },
```

```
          "username": "cwagent",
          "keystore_path": "/path/to/keystore",
          "keystore_type": "PKCS12",
          "truststore_path": "/path/to/truststore",
          "truststore_type": "PKCS12"
        },
        {
          "endpoint": "localhost:1315",
          "kafka-producer": {
            "measurement": [
              "kafka.producer.request-rate"
            ]
          },
          "append_dimensions": {
            "service.name": "kafka/1"
          }
        }
      ]
    }
  }
}
```

Collect JVM metrics

You can use the CloudWatch agent to collect Java Virtual Machine (JVM) metrics. To set this up, add a `jvm` section inside the `jmx`section of the CloudWatch agent configuration file.

The following metrics can be collected.

Metric	Dimensions	Description	
`jvm.classes.loaded`	[DEFAULT]	The total number of loaded classes. **Unit:** None **Meaningful statistics:** Minimum,	

Metric	Dimensions	Description
		Maximum, Average
`jvm.gc.collections.count`	[DEFAULT], name	The total number of garbage collections that have occurred. **Unit:** None **Meaningful statistics:** Minimum, Maximum, Average
`jvm.gc.collections.elapsed`	[DEFAULT], name	The approximate accumulated garbage collection elapsed time. **Unit:** Milliseconds **Meaningful statistics:** Minimum, Maximum, Average

Metric	Dimensions	Description
`jvm.memory.heap.init`	[DEFAULT]	The initial amount of memory that the JVM requests from the operating system for the heap. **Unit:** Bytes **Meaningful statistics:** Minimum, Maximum, Average
`jvm.memory.heap.max`	[DEFAULT]	The maximum amount of memory that can be used for the heap. **Unit:** Bytes **Meaningful statistics:** Maximum

Metric	Dimensions	Description
`jvm.memory.heap.used`	[DEFAULT]	The current heap memory usage. **Unit:** Bytes **Meaningful statistics:** Minimum, Maximum, Average
`jvm.memory.heap.committed`	[DEFAULT]	The amount of memory that is guaranteed to be available for the heap. **Unit:** Bytes **Meaningful statistics:** Minimum, Maximum, Average

Metric	Dimensions	Description
`jvm.memory.nonheap.init`	[DEFAULT]	The initial amount of memory that the JVM requests from the operating system for non-heap purposes. **Unit:** Bytes **Meaningful statistics:** Minimum, Maximum, Average
`jvm.memory.nonheap.max`	[DEFAULT]	The maximum amount of memory that can be used for non-heap purposes. **Unit:** Bytes **Meaningful statistics:** Maximum

Metric	Dimensions	Description
`jvm.memory.nonheap.used`	[DEFAULT]	The current non-heap memory usage. **Unit:** Bytes **Meaningful statistics:** Minimum, Maximum, Average
`jvm.memory.nonheap.committed`	[DEFAULT]	The amount of memory that is guaranteed to be available for non-heap purposes. **Unit:** Bytes **Meaningful statistics:** Minimum, Maximum, Average

Metric	Dimensions	Description
`jvm.memory.pool.init`	[DEFAULT], `name`	The initial amount of memory that the JVM requests from the operating system for the memory pool. **Unit:** Bytes **Meaningful statistics:** Minimum, Maximum, Average
`jvm.memory.pool.max`	[DEFAULT], `name`	The maximum amount of memory that can be used for the memory pool. **Unit:** Bytes **Meaningful statistics:** Maximum

Metric	Dimensions	Description
`jvm.memory.pool.used`	[DEFAULT], name	The current memory pool memory usage. **Unit:** Bytes **Meaningful statistics:** Minimum, Maximum, Average
`jvm.memory.pool.committed`	[DEFAULT], name	The amount of memory that is guaranteed to be available for the memory pool. **Unit:** Bytes **Meaningful statistics:** Minimum, Maximum, Average

Metric	Dimensions	Description
`jvm.threads.count`	[DEFAULT]	The current number of threads. **Unit:** None **Meaningful statistics:** Minimum, Maximum, Average

The JVM metrics are collected with the following dimensions:

Dimension	Description
[DEFAULT]	On Amazon EC2 by default, the host is also published as a dimension of metrics that are collected by the CloudWatch agent, unless you are using the `append_dimensions` field in the `metrics` section. See `omit_hostname` in the agent section of [Manually create or edit the CloudWatch agent configuration file](#) for more information. On Amazon EKS by default, k8s related context is also published as dimensions of metrics (`k8s.container.name` , `k8s.deployment.name` , `k8s.namespace.name` , `k8s.node.name` , `k8s.pod.name` , and `k8s.replicaset.name`). These can be filtered down using the `aggregation_dimensions` field.

Dimension	Description
`name`	For `jvm.gc.collections` metrics, the value is the garbage collector name. For `jvm.memory.pool` metrics, the value is the memory pool name.

Collect Kafka metrics

You can use the CloudWatch agent to collect Apache Kafka metrics. To set this up, add one or more of the following subsections inside the `jmx` section of the CloudWatch agent configuration file.

- Use a `kafka` section to collect Kafka broker metrics.
- Use a `kafka-consumer` section to collect Kafka consumer metrics.
- Use a `kafka-producer` section to collect Kafka producer metrics.

Kafka broker metrics

The following metrics can be collected for Kafka brokers.

Metric	Dimensions	Description	
`kafka.message.count`	[DEFAULT]	The number of messages received by the Kafka broker. **Unit:** None **Meaningful statistics:** Minimum, Maximum, Average	

Metric	Dimensions	Description
`kafka.request.count`	[DEFAULT], `type`	The number of requests received by the Kafka broker. **Unit:** None **Meaningful statistics:** Minimum, Maximum, Average
`kafka.request.failed`	[DEFAULT], `type`	The number of requests to the Kafka broker that resulted in a failure. **Unit:** None **Meaningful statistics:** Minimum, Maximum, Average

Metric	Dimensions	Description
`kafka.request.time.total`	[DEFAULT], `type`	The total time that the Kafka broker has taken to service requests. **Unit:** Milliseconds **Meaningful statistics:** Minimum, Maximum, Average
`kafka.request.time.50p`	[DEFAULT], `type`	The 50th percentile time that the Kafka broker has taken to service requests. **Unit:** Milliseconds **Meaningful statistics:** Minimum, Maximum, Average

Metric	Dimensions	Description
`kafka.request.time.99p`	[DEFAULT], `type`	The 99th percentile time that the Kafka broker has taken to service requests. **Unit:** Milliseconds **Meaningful statistics:** Minimum, Maximum, Average
`kafka.request.time.avg`	[DEFAULT], `type`	The average time that the Kafka broker has taken to service requests. **Unit:** Milliseconds **Meaningful statistics:** Average

Metric	Dimensions	Description
`kafka.network.io`	[DEFAULT], `state`	The number of bytes received by or sent by the Kafka broker. **Unit:** Bytes **Meaningful statistics:** Minimum, Maximum, Average
`kafka.purgatory.size`	[DEFAULT], `type`	The number of requests waiting in purgatory. **Unit:** None **Meaningful statistics:** Minimum, Maximum, Average

Metric	Dimensions	Description
kafka.partition.count	[DEFAULT]	The number of partitions on the Kafka broker. **Unit:** None **Meaningful statistics:** Minimum, Maximum, Average
kafka.partition.offline	[DEFAULT]	The number of partitions that are offline. **Unit:** None **Meaningful statistics:** Minimum, Maximum, Average
kafka.partition.under_replicated	[DEFAULT]	The number of under-replicated partitions. **Unit:** None **Meaningful statistics:** Minimum, Maximum, Average

Metric	Dimensions	Description
`kafka.isr.operation.count`	[DEFAULT], `operation`	The number of in-sync replica shrink and expand operations. **Unit:** None **Meaningful statistics:** Minimum, Maximum, Average
`kafka.max.lag`	[DEFAULT]	The maximum lag in messages between follower and leader replicas. **Unit:** None **Meaningful statistics:** Maximum

Metric	Dimensions	Description
`kafka.controller.active.count`	[DEFAULT]	The number of active controllers on the broker. **Unit:** None **Meaningful statistics:** Minimum, Maximum, Average
`kafka.leader.election.rate`	[DEFAULT]	Leader election rate. If this increases, it indicates broker failures. **Unit:** None **Meaningful statistics:** Minimum, Maximum, Average

Metric	Dimensions	Description
`kafka.unclean.election.rate`	[DEFAULT]	Unclean leader election rate. If this increases, it indicates broker failures. **Unit:** None **Meaningful statistics:** Minimum, Maximum, Average
`kafka.request.queue`	[DEFAULT]	The size of the request queue. **Unit:** None **Meaningful statistics:** Minimum, Maximum, Average
`kafka.logs.flush.time.count`	[DEFAULT]	The logs flush count. **Unit:** Milliseconds **Meaningful statistics:** Minimum, Maximum, Average

Metric	Dimensions	Description
`kafka.logs.flush.time.median`	[DEFAULT]	The 50th percentile value of the logs flush count. **Unit:** Milliseconds **Meaningful statistics:** Minimum, Maximum, Average
`kafka.logs.flush.time.99p`	[DEFAULT]	The 99th percentile value of the logs flush count. **Unit:** Milliseconds **Meaningful statistics:** Minimum, Maximum, Average

The Kafka broker metrics are collected with the following dimensions:

Dimension	Description
[DEFAULT]	On Amazon EC2 by default, the host is also published as a dimension of metrics that are

Dimension	Description
	collected by the CloudWatch agent, unless you are using the `append_dimensions` field in the `metrics` section. See `omit_hostname` in the agent section of [Manually create or edit the CloudWatch agent configuration file](#) for more information. On Amazon EKS by default, k8s related context is also published as dimensions of metrics (`k8s.container.name` , `k8s.deployment.name` , `k8s.namespace.name` , `k8s.node.name` , `k8s.pod.name` , and `k8s.replicaset.name`). These can be filtered down using the `aggregation_dimensions` field.
`type`	The request type. Possible values are `produce`, `fetch`, `fetchconsumer` , and `fetchfollower` .
`state`	The direction of network traffic. Possible values are `in` and `out`.
`operation`	The operation type for the in-sync replica. Possible values are `shrink` and `expand`.

Kafka consumer metrics

The following metrics can be collected for Kafka consumers.

Metric	Dimensions	Description
`kafka.consumer.fetch-rate`	[DEFAULT], `client-id`	The number of fetch requests

Metric	Dimensions	Description
		for all topics per second. **Unit:** None **Meaningful statistics:** Minimum, Maximum, Average
`kafka.consumer.records-lag-max`	[DEFAULT], `client-id`	The number of messages that the consumer lags behind the producer. **Unit:** None **Meaningful statistics:** Minimum, Maximum, Average
`kafka.consumer.total.bytes-consumed-rate`	[DEFAULT], `client-id`	The average number of bytes consumed for all topics per second. **Unit:** Bytes **Meaningful statistics:** Average

Metric	Dimensions	Description
`kafka.consumer.total.fetch-size-avg`	[DEFAULT], `client-id`	The number of bytes fetched per request for all topics. **Unit:** Bytes **Meaningful statistics:** Minimum, Maximum, Average
`kafka.consumer.total.records-consumed-rate`	[DEFAULT], `client-id`	The average number of records consumed for all topics per second. **Unit:** None **Meaningful statistics:** Average
`kafka.consumer.bytes-consumed-rate`	[DEFAULT], `client-id` , `topic`	The average number of bytes consumed per second. **Unit:** Bytes **Meaningful statistics:** Average

Metric	Dimensions	Description
kafka.consumer.fetch-size-avg	[DEFAULT], client-id , topic	The number of bytes fetched per request. **Unit:** Bytes **Meaningful statistics:** Minimum, Maximum, Average
kafka.consumer.records-consumed-rate	[DEFAULT], client-id , topic	The average number of records consumed per second. **Unit:** None **Meaningful statistics:** Average

The Kafka consumer metrics are collected with the following dimensions:

Dimension	Description
[DEFAULT]	On Amazon EC2 by default, the host is also published as a dimension of metrics that are collected by the CloudWatch agent, unless you are using the `append_dimensions` field in the `metrics` section. See `omit_hostname` in the agent section of [Manually create or edit the CloudWatch agent configuration file](#) for more information.

Dimension	Description
	On Amazon EKS by default, k8s related context is also published as dimensions of metrics (`k8s.container.name` , `k8s.deployment.name` , `k8s.namespace.name` , `k8s.node.name` , `k8s.pod.name` , and `k8s.replicaset.name`). These can be filtered down using the `aggregation_dimensions` field.
`client-id`	The ID of the client.
`topic`	The Kafka topic.

Kafka producer metrics

The following metrics can be collected for Kafka producers.

Metric	Dimensions	Description
`kafka.producer.io-wait-time-ns-avg`	[DEFAULT], `client-id`	The average length of time the I/O thread spent waiting for a socket ready for reads or writes. **Unit:** None **Meaningful statistics:** Average
`kafka.producer.out`	[DEFAULT], `client-id`	The average number of

Metric	Dimensions	Description
going-byte-rate		outgoing bytes sent per second to all servers. **Unit:** Bytes **Meaningful statistics:** Average
kafka.producer.request-latency-avg	[DEFAULT], client-id	The average request latency. **Unit:** Milliseconds **Meaningful statistics:** Average
kafka.producer.request-rate	[DEFAULT], client-id	The average number of requests sent per second. **Unit:** None **Meaningful statistics:** Average

Metric	Dimensions	Description
`kafka.producer.response-rate`	[DEFAULT], `client-id`	The number of responses received per second. **Unit:** None **Meaningful statistics:** Minimum, Maximum, Average
`kafka.producer.byte-rate`	[DEFAULT], `client-id` , `topic`	The average number of bytes sent per second for a topic. **Unit:** Bytes **Meaningful statistics:** Average
`kafka.producer.compression-rate`	[DEFAULT], `client-id` , `topic`	The average compression rate of record batches for a topic. **Unit:** None **Meaningful statistics:** Average

Metric	Dimensions	Description
kafka.producer.record-error-rate	[DEFAULT], client-id , topic	The average per-second number of record sends that resulted in errors for a topic. **Unit:** None **Meaningful statistics:** Average
kafka.producer.record-retry-rate	[DEFAULT], client-id , topic	The average per-second number of retried record sends for a topic. **Unit:** None **Meaningful statistics:** Average

Metric	Dimensions	Description
`kafka.producer.record-send-rate`	[DEFAULT], `client-id` , `topic`	The average number of records sent per second for a topic. **Unit:** None **Meaningful statistics:** Average

Kafka producer metrics are collected with the following dimensions:

Dimension	Description
[DEFAULT]	On Amazon EC2 by default, the host is also published as a dimension of metrics that are collected by the CloudWatch agent, unless you are using the `append_dimensions` field in the `metrics` section. See `omit_hostname` in the agent section of [Manually create or edit the CloudWatch agent configuration file]() for more information. On Amazon EKS by default, k8s related context is also published as dimensions of metrics (`k8s.container.name` , `k8s.deployment.name` , `k8s.namespace.name` , `k8s.node.name` , `k8s.pod.name` , and `k8s.replicaset.name`). These can be filtered down using the `aggregation_dimensions` field.

Dimension	Description
client-id	The ID of the client.
topic	The Kafka topic.

Collect Tomcat metrics

You can use the CloudWatch agent to collect Apache Tomcat metrics. To set this up, add a `tomcat` section inside the `metrics_collected` section of the CloudWatch agent configuration file.

The following metrics can be collected.

Metric	Dimensions	Description
`tomcat.sessions`	[DEFAULT]	The number of active sessions. **Unit:** None **Meaningful statistics:** Minimum, Maximum, Average
`tomcat.errors`	[DEFAULT], `proto_handler`	The number of errors encountered. **Unit:** None **Meaningful statistics:** Minimum, Maximum, Average

Metric	Dimensions	Description
tomcat.processing_time	[DEFAULT], proto_handler	The total processing time. **Unit:** Milliseconds **Meaningful statistics:** Minimum, Maximum, Average
tomcat.traffic	[DEFAULT], proto_handler	The number of bytes received and sent. **Unit:** Bytes **Meaningful statistics:** Minimum, Maximum, Average
tomcat.threads	[DEFAULT], proto_handler	The number of threads. **Unit:** None **Meaningful statistics:** Minimum, Maximum, Average

Metric	Dimensions	Description
tomcat.max_time	[DEFAULT], proto_handler , direction	Maximum time to process a request. **Unit:** Milliseconds **Meaningful statistics:** Maximum
tomcat.request_count	[DEFAULT], proto_handler	The total requests. **Unit:** None **Meaningful statistics:** Minimum, Maximum, Average

Tomcat metrics are collected with the following dimensions:

Dimension	Description
[DEFAULT]	On Amazon EC2 by default, the host is also published as a dimension of metrics that are collected by the CloudWatch agent, unless you are using the `append_dimensions` field in the `metrics` section. See `omit_hostname` in the agent section of [Manually create or edit the CloudWatch agent configuration file](#) for more information.

Dimension	Description
	On Amazon EKS by default, k8s related context is also published as dimensions of metrics (`k8s.container.name` , `k8s.deployment.name` , `k8s.namespace.name` , `k8s.node.name` , `k8s.pod.name` , and `k8s.replicaset.name`). These can be filtered down using the `aggregation_dimensions` field.
`proto_handler`	The `proto_handler` is an identifier for a connector, which is provided in the `<protocol>-<type>-<port>` format (for example, `http-nio-8080`).
`direction`	The traffic direction. Possible values are `received` and `sent`.

Collect metrics and traces with OpenTelemetry

You can collect metrics and traces from your applications or services using the CloudWatch agent with the OpenTelemetry Protocol (OTLP), which is a popular open source solution. You can use any OpenTelemetry SDK to send metrics and traces to the CloudWatch agent. For more information about the available OpenTelemetry SDKs, see [OpenTelemetry Supported Language APIs & SDKs.](#).

To collect OpenTelemetry metrics and traces, add an `otlp` section to the CloudWatch agent configuration file. The section has the following fields:

- `grpc_endpoint` – Optional. Specifies the address for the CloudWatch agent to use to listen for OpenTelemetry metrics or traces sent using gRPC Remote Procedure Calls. The format is `ip:port`. This address must match the address set for the gRPC exporter in the OpenTelemetry SDK. If you omit this field, the default of `127.0.0.1:4317` is used.
- `http_endpoint` – Optional. Specifies the address for the CloudWatch agent to use to listen for OpenTelemetry metrics or traces sent over HTTP. The format is `ip:port`. This address must

match the address set for the HTTP exporter in the OpenTelemetry SDK. If you omit this field, the default of `127.0.0.1:4318` is used.

- `tls` – Optional. Specifies that the server should be configured with TLS.
  - `cert_file` – Path to the TLS certificate to use for TLS required connections.
  - `key_file` – Path to the TLS key to use for TLS required connections.

The `otlp` section can be placed in multiple sections within the CloudWatch agent configuration file depending on how and where you want to send the metrics and traces. Each `otlp` section requires a unique endpoint. For detailed information about splitting the metrics and traces endpoints, see OTLP Exporter Configuration in the OpenTelemetry SDK documentation.

To send metrics to CloudWatch or Amazon Managed Service for Prometheus, add the `otlp` section under `metrics_collected` within the `metrics` section. For more information about sending metrics to different destinations, see Manually create or edit the CloudWatch agent configuration file. The following example shows a configuration that sends metrics to CloudWatch:

```
{
  "metrics": {
    "metrics_collected": {
      "otlp": {
        "grpc_endpoint": "127.0.0.1:4317",
        "http_endpoint": "127.0.0.1:4318"
      }
    }
  }
}
```

To send metrics to Amazon CloudWatch Logs using the Embedded metric format (EMF), add the `otlp` section under `metrics_collected` within the `logs` section. This sends the EMF logs by default to the `/aws/cwagent` log group and a generated log stream. The metrics are extracted into the `CWAgent` namespace by default. The following example shows a configuration that sends metrics as EMF logs to CloudWatch Logs:

```
{
  "logs": {
    "metrics_collected": {
      "otlp": {
```

```
          "grpc_endpoint": "127.0.0.1:4317",
          "http_endpoint": "127.0.0.1:4318"
        }
      }
    }
}
```

To send traces to Amazon X-Ray, add the `otlp` section under `traces_collected` within the `traces` section. The following example shows a configuration that sends traces to X-Ray:

```
{
  "traces": {
    "traces_collected": {
      "otlp": {
        "grpc_endpoint": "127.0.0.1:4317",
        "http_endpoint": "127.0.0.1:4318"
      }
    }
  }
}
```

Collect process metrics with the procstat plugin

The *procstat* plugin enables you to collect metrics from individual processes. The plugin is supported on Linux servers and on servers running supported version of Windows Server. This section describes how to configure the CloudWatch agent for procstat and view metrics the CloudWatch agent imports. It also lists the metrics that procstat collects.

> **Note**
>
> The `procstat` plugin is not supported for the Fargate launch type in Amazon ECS environments.

Topics

- Viewing process metrics imported by the CloudWatch agent

Configure the CloudWatch agent for procstat

To use the procstat plugin, add a `procstat` section in the `metrics_collected` section of the CloudWatch agent configuration file. There are three ways to specify the processes to monitor. You can use only one of these methods, but you can use that method to specify one or more processes to monitor.

- `pid_file`: Selects processes by the names of the process identification number (PID) files they create.
- `exe`: Selects the processes that have process names that match the string that you specify, using regular expression matching rules. The match is a "contains" match, meaning that if you specify `agent` as the term to match, processes with names like `cloudwatchagent` match the term. For more information, see Syntax.
- `pattern`: Selects processes by the command lines used to start the processes. All processes are selected that have command lines matching the specified string using regular expression matching rules. The entire command line is checked, including parameters and options used with the command.

  The match is a "contains" match, meaning that if you specify `-c` as the term to match, processes with parameters like `-config` match the term.

The CloudWatch agent uses only one of these methods, even if you include more than one of the above sections. If you specify more than one section, the CloudWatch agent uses the `pid_file` section if it is present. If not, it uses the `exe` section.

On Linux servers, the strings that you specify in an `exe` or `pattern` section are evaluated as regular expressions. On servers running Windows Server, these strings are evaluated as WMI queries. An example would be `pattern: "%apache%"`. For more information, see LIKE Operator.

Whichever method you use, you can include an optional `metrics_collection_interval` parameter, which specifies how often in seconds to collect those metrics. If you omit this parameter, the default value of 60 seconds is used.

In the examples in the following sections, the `procstat` section is the only section included in the `metrics_collected` section of the agent configuration file. Actual configuration files can also include other sections in `metrics_collected`. For more information, see [Manually create or edit the CloudWatch agent configuration file](#).

Configure with pid_file

The following example `procstat` section monitors the processes that create the PID files `example1.pid` and `example2.pid`. Different metrics are collected from each process. Metrics collected from the process that creates `example2.pid` are collected every 10 seconds, and the metrics collected from the `example1.pid` process are collected every 60 seconds, the default value.

```
{
    "metrics": {
        "metrics_collected": {
            "procstat": [
                {
                    "pid_file": "/var/run/example1.pid",
                    "measurement": [
                        "cpu_usage",
                        "memory_rss"
                    ]
                },
                {
                    "pid_file": "/var/run/example2.pid",
                    "measurement": [
                        "read_bytes",
                        "read_count",
                        "write_bytes"
                    ],
                    "metrics_collection_interval": 10
                }
            ]
        }
    }
}
```

Configuring with exe

The following example `procstat` section monitors all processes with names that match the strings `agent` or `plugin`. The same metrics are collected from each process.

```
{
```

```
    "metrics": {
        "metrics_collected": {
            "procstat": [
                {
                    "exe": "agent",
                    "measurement": [
                        "cpu_time",
                        "cpu_time_system",
                        "cpu_time_user"
                    ]
                },
                {
                    "exe": "plugin",
                    "measurement": [
                        "cpu_time",
                        "cpu_time_system",
                        "cpu_time_user"
                    ]
                }
            ]
        }
    }
}
```

Configuring with pattern

The following example `procstat` section monitors all processes with command lines that match the strings `config` or `-c`. The same metrics are collected from each process.

```
{
    "metrics": {
        "metrics_collected": {
            "procstat": [
                {
                    "pattern": "config",
                    "measurement": [
                        "rlimit_memory_data_hard",
                        "rlimit_memory_data_soft",
                        "rlimit_memory_stack_hard",
                        "rlimit_memory_stack_soft"
                    ]
                },
                {
```

```
                    "pattern": "-c",
                    "measurement": [
                        "rlimit_memory_data_hard",
                        "rlimit_memory_data_soft",
                        "rlimit_memory_stack_hard",
                        "rlimit_memory_stack_soft"
                    ]
                }
            ]
        }
    }
}
```

Metrics collected by procstat

The following table lists the metrics that you can collect with the `procstat` plugin.

The CloudWatch agent adds `procstat` to the beginning of the following metric names. There is a different syntax depending on whether it was collected from a Linux server or a server running Windows Server. For example, the `cpu_time` metric appears as `procstat_cpu_time` when collected from Linux and as `procstat cpu_time` when collected from Windows Server.

Metric name	Available on	Description
`cpu_time`	Linux	The amount of time that the process uses the CPU. This metric is measured in hundredths of a second. Unit: Count
`cpu_time_guest`	Linux	The amount of time that the process is in guest mode. This metric is

Metric name	Available on	Description
		measured in hundredths of a second. Type: Float Unit: None
`cpu_time_guest_nice`	Linux	The amount of time that the process is running in a nice guest. This metric is measured in hundredths of a second. Type: Float Unit: None
`cpu_time_idle`	Linux	The amount of time that the process is in idle mode. This metric is measured in hundredths of a second. Type: Float Unit: None

Metric name	Available on	Description
cpu_time_iowait	Linux	The amount of time that the process is waiting for I/O operations to complete. This metric is measured in hundredths of a second. Type: Float Unit: None
cpu_time_irq	Linux	The amount of time that the process is servicing interrupts. This metric is measured in hundredths of a second. Type: Float Unit: None

Metric name	Available on	Description
`cpu_time_nice`	Linux	The amount of time that the process is in nice mode. This metric is measured in hundredths of a second. Type: Float Unit: None
`cpu_time_soft_irq`	Linux	The amount of time that the process is servicing software interrupts. This metric is measured in hundredths of a second. Type: Float Unit: None

Metric name	Available on	Description
`cpu_time_steal`	Linux	The amount of time spent running in other operating systems when running in a virtualized environment. This metric is measured in hundredths of a second. Type: Float Unit: None
`cpu_time_stolen`	Linux, Windows Server	The amount of time that the process is in *stolen time*, which is time spent in other operating systems in a virtualized environment. This metric is measured in hundredths of a second. Type: Float Unit: None

Metric name	Available on	Description
`cpu_time_system`	Linux, Windows Server, macOS	The amount of time that the process is in system mode. This metric is measured in hundredths of a second. Type: Float Unit: Count
`cpu_time_user`	Linux, Windows Server, macOS	The amount of time that the process is in user mode. This metric is measured in hundredths of a second. Unit: Count
`cpu_usage`	Linux, Windows Server, macOS	The percentage of time that the process is active in any capacity. Unit: Percent

Metric name	Available on	Description
`memory_data`	Linux, macOS	The amount of memory that the process uses for data. Unit: Bytes
`memory_locked`	Linux, macOS	The amount of memory that the process has locked. Unit: Bytes
`memory_rss`	Linux, Windows Server, macOS	The amount of real memory (resident set) that the process is using. Unit: Bytes
`memory_stack`	Linux, macOS	The amount of stack memory that the process is using. Unit: Bytes

Metric name	Available on	Description
`memory_swap`	Linux, macOS	The amount of swap memory that the process is using. Unit: Bytes
`memory_vms`	Linux, Windows Server, macOS	The amount of virtual memory that the process is using. Unit: Bytes
`num_fds`	Linux	The number of file descripto rs that this process has open. Unit: None
`num_threads`	Linux, Windows, macOS	The number of threads in this process. Unit: None
`pid`	Linux, Windows Server, macOS	Process identifier (ID). Unit: None

Metric name	Available on	Description
`pid_count`	Linux, Windows Server. macOS	The number of process IDs associated with the process. On Linux servers and macOS computers the full name of this metric is `procstat_lookup_pid_count` and on Windows Server it is `procstat_lookup pid_count` . Unit: None
`read_bytes`	Linux, Windows Server	The number of bytes that the process has read from disks. Unit: Bytes

Metric name	Available on	Description
`write_bytes`	Linux, Windows Server	The number of bytes that the process has written to disks. Unit: Bytes
`read_count`	Linux, Windows Server	The number of disk read operation s that the process has executed. Unit: None
`rlimit_realtime_pr iority_hard`	Linux	The hard limit on the real-time priority that can be set for this process. Unit: None
`rlimit_realtime_pr iority_soft`	Linux	The soft limit on the real-time priority that can be set for this process. Unit: None

Metric name	Available on	Description
rlimit_signals_pending_hard	Linux	The hard limit on maximum number of signals that can be queued by this process. Unit: None
rlimit_signals_pending_soft	Linux	The soft limit on maximum number of signals that can be queued by this process. Unit: None
rlimit_nice_priority_hard	Linux	The hard limit on the maximum nice priority that can be set by this process. Unit: None

Metric name	Available on	Description
`rlimit_nice_priority_soft`	Linux	The soft limit on the maximum nice priority that can be set by this process. Unit: None
`rlimit_num_fds_hard`	Linux	The hard limit on the maximum number of file descriptors that this process can have open. Unit: None
`rlimit_num_fds_soft`	Linux	The soft limit on the maximum number of file descriptors that this process can have open. Unit: None

Metric name	Available on	Description
`write_count`	Linux, Windows Server	The number of disk write operations that the process has executed. Unit: None
`involuntary_context_switches`	Linux	The number of times that the process was involuntarily context-switched. Unit: None
`voluntary_context_switches`	Linux	The number of times that the process was context-switched voluntarily. Unit: None
`realtime_priority`	Linux	The current usage of real-time priority for the process. Unit: None

Metric name	Available on	Description
nice_priority	Linux	The current usage of nice priority for the process. Unit: None
signals_pending	Linux	The number of signals pending to be handled by the process. Unit: None
rlimit_cpu_time_hard	Linux	The hard CPU time resource limit for the process. Unit: None
rlimit_cpu_time_soft	Linux	The soft CPU time resource limit for the process. Unit: None
rlimit_file_locks_hard	Linux	The hard file locks resource limit for the process. Unit: None

Metric name	Available on	Description
`rlimit_file_locks_soft`	Linux	The soft file locks resource limit for the process. Unit: None
`rlimit_memory_data_hard`	Linux	The hard resource limit on the process for memory used for data. Unit: Bytes
`rlimit_memory_data_soft`	Linux	The soft resource limit on the process for memory used for data. Unit: Bytes
`rlimit_memory_lock ed_hard`	Linux	The hard resource limit on the process for locked memory. Unit: Bytes

Metric name	Available on	Description
rlimit_memory_locked_soft	Linux	The soft resource limit on the process for locked memory. Unit: Bytes
rlimit_memory_rss_hard	Linux	The hard resource limit on the process for physical memory. Unit: Bytes
rlimit_memory_rss_soft	Linux	The soft resource limit on the process for physical memory. Unit: Bytes
rlimit_memory_stack_hard	Linux	The hard resource limit on the process stack. Unit: Bytes
rlimit_memory_stack_soft	Linux	The soft resource limit on the process stack. Unit: Bytes

Metric name	Available on	Description
`rlimit_memory_vms_hard`	Linux	The hard resource limit on the process for virtual memory. Unit: Bytes
`rlimit_memory_vms_soft`	Linux	The soft resource limit on the process for virtual memory. Unit: Bytes

Viewing process metrics imported by the CloudWatch agent

After importing process metrics into CloudWatch, you can view these metrics as time series graphs, and create alarms that can watch these metrics and notify you if they breach a threshold that you specify. The following procedure shows how to view process metrics as a time series graph. For more information about setting alarms, see [Using Amazon CloudWatch alarms](#).

To view process metrics in the CloudWatch console

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Metrics**.
3. Choose the namespace for the metrics collected by the agent. By default, this is **CWAgent**, but you may have specified a different namespace in the CloudWatch agent configuration file.
4. Choose a metric dimension (for example, **Per-Instance Metrics**).
5. The **All metrics** tab displays all metrics for that dimension in the namespace. You can do the following:

    a. To graph a metric, select the check box next to the metric. To select all metrics, select the check box in the heading row of the table.

b. To sort the table, use the column heading.

c. To filter by resource, choose the resource ID and then choose **Add to search**.

d. To filter by metric, choose the metric name and then choose **Add to search**.

6. (Optional) To add this graph to a CloudWatch dashboard, choose **Actions**, **Add to dashboard**.

Retrieve custom metrics with StatsD

You can retrieve additional custom metrics from your applications or services using the CloudWatch agent with the `StatsD` protocol. StatsD is a popular open-source solution that can gather metrics from a wide variety of applications. StatsD is especially useful for instrumenting your own metrics. For an example of using the CloudWatch agent and StatsD together, see [How to better monitor your custom application metrics using Amazon CloudWatch Agent]().

`StatsD` is supported on both Linux servers and servers running Windows Server. CloudWatch supports the following `StatsD` format:

```
MetricName:value|type|@sample_rate|#tag1:
  value,tag1...
```

- `MetricName` – A string with no colons, bars, # characters, or @ characters.
- `value` – This can be either integer or float.
- `type` – Specify `c` for counter, `g` for gauge, `ms` for timer, `h` for histogram, or `s` for set.
- `sample_rate` – (Optional) A float between 0 and 1, inclusive. Use only for counter, histogram, and timer metrics. The default value is 1 (sampling 100% of the time).
- `tags` – (Optional) A comma-separated list of tags. `StatsD` tags are similar to dimensions in CloudWatch. Use colons for key/value tags, such as `env:prod`.

You can use any `StatsD` client that follows this format to send the metrics to the CloudWatch agent. For more information about some of the available `StatsD` clients, see the StatsD client page on GitHub.

To collect these custom metrics, add a `"statsd": {}` line to the `metrics_collected` section of the agent configuration file. You can add this line manually. If you use the wizard to create the configuration file, it's done for you. For more information, see Create the CloudWatch agent configuration file.

The StatsD default configuration works for most users. There are optional fields that you can add to the **statsd** section of the agent configuration file as needed:

- service_address – The service address to which the CloudWatch agent should listen. The format is *ip*:*port*. If you omit the IP address, the agent listens on all available interfaces. Only the UDP format is supported, so you don't need to specify a UDP prefix.

  The default value is :8125.
- metrics_collection_interval – How often in seconds that the StatsD plugin runs and collects metrics. The default value is 10 seconds. The range is 1–172,000.
- metrics_aggregation_interval – How often in seconds CloudWatch aggregates metrics into single data points. The default value is 60 seconds.

  For example, if metrics_collection_interval is 10 and metrics_aggregation_interval is 60, CloudWatch collects data every 10 seconds. After each minute, the six data readings from that minute are aggregated into a single data point, which is sent to CloudWatch.

  The range is 0–172,000. Setting metrics_aggregation_interval to 0 disables the aggregation of StatsD metrics.
- allowed_pending_messages – The number of UDP messages that are allowed to queue up. When the queue is full, the StatsD server starts dropping packets. The default value is 10000.
- drop_original_metrics – Optional. If you are using the aggregation_dimensions field in the metrics section to roll up metrics into aggregated results, then by default the agent sends both the aggregated metrics and the original metrics that are separated for each value of the dimension. If you don't want the original metrics to be sent to CloudWatch, you can specify this parameter with a list of metrics. The metrics specified along with this parameter don't have their metrics by dimension reported to CloudWatch. Instead, only the aggregated metrics are reported. This reduces the number of metrics that the agent collects, reducing your costs.

The following is an example of the **statsd** section of the agent configuration file, using the default port and custom collection and aggregation intervals.

```
{
   "metrics":{
      "metrics_collected":{
         "statsd":{
```

```
                "service_address":":8125",
                "metrics_collection_interval":60,
                "metrics_aggregation_interval":300
            }
        }
    }
}
```

Viewing StatsD metrics imported by the CloudWatch agent

After importing StatsD metrics into CloudWatch, you can view these metrics as time series graphs, and create alarms that can watch these metrics and notify you if they breach a threshold that you specify. The following procedure shows how to view StatsD metrics as a time series graph. For more information about setting alarms, see Using Amazon CloudWatch alarms.

To view StatsD metrics in the CloudWatch console

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Metrics**.
3. Choose the namespace for the metrics collected by the agent. By default, this is **CWAgent**, but you may have specified a different namespace in the CloudWatch agent configuration file.
4. Choose a metric dimension (for example, **Per-Instance Metrics**).
5. The **All metrics** tab displays all metrics for that dimension in the namespace. You can do the following:
   a. To graph a metric, select the check box next to the metric. To select all metrics, select the check box in the heading row of the table.
   b. To sort the table, use the column heading.
   c. To filter by resource, choose the resource ID and then choose **Add to search**.
   d. To filter by metric, choose the metric name and then choose **Add to search**.
6. (Optional) To add this graph to a CloudWatch dashboard, choose **Actions**, **Add to dashboard**.

Retrieve custom metrics with collectd

You can retrieve additional metrics from your applications or services using the CloudWatch agent with the collectd protocol, which is supported only on Linux servers. collectd is a popular open-source solution with plugins that can gather system statistics for a wide variety of applications. By combining the system metrics that the CloudWatch agent can already collect with the additional metrics from collectd, you can better monitor, analyze, and troubleshoot your systems and applications. For more information about collectd, see [collectd - The system statistics collection daemon](#).

You use the collectd software to send the metrics to the CloudWatch agent. For the collectd metrics, the CloudWatch agent acts as the server while the collectd plugin acts as the client.

The collectd software is not installed automatically on every server. On a server running Amazon Linux 2, follow these steps to install collectd

```
sudo amazon-linux-extras install collectd
```

For information about installing collectd on other systems, see the [Download page for collectd.](#)

To collect these custom metrics, add a **"collectd": {}** line to the **metrics_collected** section of the agent configuration file. You can add this line manually. If you use the wizard to create the configuration file, it is done for you. For more information, see [Create the CloudWatch agent configuration file](#).

Optional parameters are also available. If you are using collectd and you do not use `/etc/collectd/auth_file` as your **collectd_auth_file**, you must set some of these options.

- **service_address:** The service address to which the CloudWatch agent should listen. The format is `"udp://ip:port`. The default is `udp://127.0.0.1:25826`.
- **name_prefix:** A prefix to attach to the beginning of the name of each collectd metric. The default is `collectd_`. The maximum length is 255 characters.
- **collectd_security_level:** Sets the security level for network communication. The default is **encrypt**.

  encrypt specifies that only encrypted data is accepted. **sign** specifies that only signed and encrypted data is accepted. **none** specifies that all data is accepted. If you specify a value for **collectd_auth_file**, encrypted data is decrypted if possible.

  For more information, see [Client setup](#) and [Possible interactions](#) in the collectd Wiki.
- **collectd_auth_file** Sets a file in which user names are mapped to passwords. These passwords are used to verify signatures and to decrypt encrypted network packets. If given, signed data is verified and encrypted packets are decrypted. Otherwise, signed data is accepted without checking the signature and encrypted data cannot be decrypted.

The default is `/etc/collectd/auth_file`.

If **collectd_security_level** is set to **none**, this is optional. If you set **collectd_security_level** to `encrypt` or **sign**, you must specify **collectd_auth_file**.

For the format of the auth file, each line is a user name followed by a colon and any number of spaces followed by the password. For example:

`user1: user1_password`

`user2: user2_password`

- **collectd_typesdb:** A list of one or more files that contain the dataset descriptions. The list must be surrounded by brackets, even if there is just one entry in the list. Each entry in the list must be surrounded by double quotes. If there are multiple entries, separate them with commas. The default on Linux servers is `["/usr/share/collectd/types.db"]`. The default on macOs computers depends on the version of collectd. For example, `["/usr/local/Cellar/collectd/5.12.0/share/collectd/types.db"]`.

  For more information, see https://www.collectd.org/documentation/manpages/types.db.html.
- **metrics_aggregation_interval:** How often in seconds CloudWatch aggregates metrics into single data points. The default is 60 seconds. The range is 0 to 172,000. Setting it to 0 disables the aggregation of collectd metrics.

The following is an example of the collectd section of the agent configuration file.

```
{
   "metrics":{
      "metrics_collected":{
         "collectd":{
            "name_prefix":"My_collectd_metrics_",
            "metrics_aggregation_interval":120
         }
      }
   }
}
```

Viewing collectd metrics imported by the CloudWatch agent

After importing collectd metrics into CloudWatch, you can view these metrics as time series graphs, and create alarms that can watch these metrics and notify you if they breach a threshold that you specify. The following procedure shows how to view collectd metrics as a time series graph. For more information about setting alarms, see [Using Amazon CloudWatch alarms](#).

To view collectd metrics in the CloudWatch console

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Metrics**.
3. Choose the namespace for the metrics collected by the agent. By default, this is **CWAgent**, but you may have specified a different namespace in the CloudWatch agent configuration file.
4. Choose a metric dimension (for example, **Per-Instance Metrics**).
5. The **All metrics** tab displays all metrics for that dimension in the namespace. You can do the following:

   a. To graph a metric, select the check box next to the metric. To select all metrics, select the check box in the heading row of the table.
   b. To sort the table, use the column heading.
   c. To filter by resource, choose the resource ID and then choose **Add to search**.
   d. To filter by metric, choose the metric name and then choose **Add to search**.
6. (Optional) To add this graph to a CloudWatch dashboard, choose **Actions**, **Add to dashboard**.

Set up and configure Prometheus metrics collection on Amazon EC2 instances

The following sections explain how to install the CloudWatch agent with Prometheus monitoring on EC2 instances, and how to configure the agent to scrape additional targets. It also provides tutorials for setting up sample workloads to use testing with Prometheus monitoring.

Both Linux and Windows instances are supported.

For information about the operating systems supported by the CloudWatch agent, see [Collect metrics, logs, and traces with the CloudWatch agent](#)

VPC security group requirements

If you are using a VPC, the following requirements apply.

- The ingress rules of the security groups for the Prometheus workloads must open the Prometheus ports to the CloudWatch agent for scraping the Prometheus metrics by the private IP.
- The egress rules of the security group for the CloudWatch agent must allow the CloudWatch agent to connect to the Prometheus workloads' port by private IP.

Topics

Step 1: Install the CloudWatch agent

The first step is to install the CloudWatch agent on the EC2 instance. For instructions, see Install the CloudWatch agent.

Step 2: Scrape Prometheus sources and import metrics

The CloudWatch agent with Prometheus monitoring needs two configurations to scrape the Prometheus metrics. One is for the standard Prometheus configurations as documented in <scrape_config>in the Prometheus documentation. The other is for the CloudWatch agent configuration.

Prometheus scrape configuration

The CloudWatch agent supports the standard Prometheus scrape configurations as documented in <scrape_config> in the Prometheus documentation. You can edit this section to update the configurations that are already in this file, and add additional Prometheus scraping targets. A sample configuration file contains the following global configuration lines:

```
PS C:\ProgramData\Amazon\AmazonCloudWatchAgent> cat prometheus.yaml
global:
  scrape_interval: 1m
  scrape_timeout: 10s
scrape_configs:
- job_name: MY_JOB
  sample_limit: 10000
  file_sd_configs:
```

```
    - files: ["C:\\ProgramData\\Amazon\\AmazonCloudWatchAgent\\prometheus_sd_1.yaml",
 "C:\\ProgramData\\Amazon\\AmazonCloudWatchAgent\\prometheus_sd_2.yaml"]
```

The `global` section specifies parameters that are valid in all configuration contexts. They also serve as defaults for other configuration sections. It contains the following parameters:

- `scrape_interval`— Defines how frequently to scrape targets.
- `scrape_timeout`— Defines how long to wait before a scrape request times out.

The `scrape_configs` section specifies a set of targets and parameters that define how to scrape them. It contains the following parameters:

- `job_name`— The job name assigned to scraped metrics by default.
- `sample_limit`— Per-scrape limit on the number of scraped samples that will be accepted.
- `file_sd_configs`— List of file service discovery configurations. It reads a set of files containing a list of zero or more static configs. The `file_sd_configs` section contains a `files` parameter which defines patterns for files from which target groups are extracted.

The CloudWatch agent supports the following service discovery configuration types.

`static_config` Allows specifying a list of targets and a common label set for them. It is the canonical way to specify static targets in a scrape configuration.

The following is a sample static config to scrape Prometheus metrics from a local host. Metrics can also be scraped from other servers if the Prometheus port is open to the server where the agent runs.

```
PS C:\ProgramData\Amazon\AmazonCloudWatchAgent> cat prometheus_sd_1.yaml
- targets:
    - 127.0.0.1:9404
  labels:
    key1: value1
    key2: value2
```

This example contains the following parameters:

- `targets`— The targets scraped by the static config.
- `labels`— Labels assigned to all metrics that are scraped from the targets.

`ec2_sd_config` Allows retrieving scrape targets from Amazon EC2 instances. The following is a sample `ec2_sd_config` to scrape Prometheus metrics from a list of EC2 instances. The Prometheus ports of these instances have to open to the server where the CloudWatch agent runs. The IAM role for the EC2 instance where the CloudWatch agent runs must include the `ec2:DescribeInstance` permission. For example, you could attach the managed policy **AmazonEC2ReadOnlyAccess** to the instance running the CloudWatch agent.

```
PS C:\ProgramData\Amazon\AmazonCloudWatchAgent> cat prometheus.yaml
global:
  scrape_interval: 1m
  scrape_timeout: 10s
scrape_configs:
  - job_name: MY_JOB
    sample_limit: 10000
    ec2_sd_configs:
      - region: us-east-1
        port: 9404
        filters:
          - name: instance-id
            values:
              - i-98765432109876543
              - i-12345678901234567
```

This example contains the following parameters:

- `region`— The Amazon Region where the target EC2 instance is. If you leave this blank, the Region from the instance metadata is used.
- `port`— The port to scrape metrics from.
- `filters`— Optional filters to use to filter the instance list. This example filters based on EC2 instance IDs. For more criteria that you can filter on, see [DescribeInstances](#).

CloudWatch agent configuration for Prometheus

The CloudWatch agent configuration file includes `prometheus` sections under both `logs` and `metrics_collected`. It includes the following parameters.

- **cluster_name**— specifies the cluster name to be added as a label in the log event. This field is optional.
- **log_group_name**— specifies the log group name for the scraped Prometheus metrics.

- **prometheus_config_path**— specifies the Prometheus scrape configuration file path.
- **emf_processor**— specifies the embedded metric format processor configuration. For more information about embedded metric format, see Embedding metrics within logs.

  The `emf_processor` section can contain the following parameters:

  - **metric_declaration_dedup**— It set to true, the de-duplication function for the embedded metric format metrics is enabled.
  - **metric_namespace**— Specifies the metric namespace for the emitted CloudWatch metrics.
  - **metric_unit**— Specifies the metric name:metric unit map. For information about supported metric units, see MetricDatum.
  - **metric_declaration**— are sections that specify the array of logs with embedded metric format to be generated. There are `metric_declaration` sections for each Prometheus source that the CloudWatch agent imports from by default. These sections each include the following fields:
    - `source_labels` specifies the value of the labels that are checked by the `label_matcher` line.
    - `label_matcher` is a regular expression that checks the value of the labels listed in `source_labels`. The metrics that match are enabled for inclusion in the embedded metric format sent to CloudWatch.
    - `metric_selectors` is a regular expression that specifies the metrics to be collected and sent to CloudWatch.
    - `dimensions` is the list of labels to be used as CloudWatch dimensions for each selected metric.

The following is an example CloudWatch agent configuration for Prometheus.

```
{
   "logs":{
      "metrics_collected":{
         "prometheus":{
            "cluster_name":"prometheus-cluster",
            "log_group_name":"Prometheus",
            "prometheus_config_path":"C:\\ProgramData\\Amazon\\AmazonCloudWatchAgent\
\prometheus.yaml",
            "emf_processor":{
               "metric_declaration_dedup":true,
               "metric_namespace":"CWAgent-Prometheus",
```

```
                "metric_unit":{
                   "jvm_threads_current": "Count",
                   "jvm_gc_collection_seconds_sum": "Milliseconds"
                },
                "metric_declaration":[
                   {
                      "source_labels":[
                         "job", "key2"
                      ],
                      "label_matcher":"MY_JOB;^value2",
                      "dimensions":[
                         [
                            "key1", "key2"
                         ],
                         [
                            "key2"
                         ]
                      ],
                      "metric_selectors":[
                         "^jvm_threads_current$",
                         "^jvm_gc_collection_seconds_sum$"
                      ]
                   }
                ]
             }
          }
       }
    }
}
```

The previous example configures an embedded metric format section to be sent as a log event if the following conditions are met:

- The value of the label `job` is `MY_JOB`
- The value of the label `key2` is `value2`
- The Prometheus metrics `jvm_threads_current` and `jvm_gc_collection_seconds_sum` contains both `job` and `key2` labels.

The log event that is sent includes the following highlighted section.

```
{
```

```
    "CloudWatchMetrics": [
        {
            "Metrics": [
                {
                    "Unit": "Count",
                    "Name": "jvm_threads_current"
                },
                {
                    "Unit": "Milliseconds",
                    "Name": "jvm_gc_collection_seconds_sum"
                }
            ],
            "Dimensions": [
                [
                    "key1",
                    "key2"
                ],
                [
                    "key2"
                ]
            ],
            "Namespace": "CWAgent-Prometheus"
        }
    ],
    "ClusterName": "prometheus-cluster",
    "InstanceId": "i-0e45bd06f196096c8",
    "Timestamp": "1607966368109",
    "Version": "0",
    "host": "EC2AMAZ-PDDOIUM",
    "instance": "127.0.0.1:9404",
    "jvm_threads_current": 2,
    "jvm_gc_collection_seconds_sum": 0.006000000000000002,
    "prom_metric_type": "gauge",
    ...
}
```

Example: Set up Java/JMX sample workloads for Prometheus metric testing

JMX Exporter is an official Prometheus exporter that can scrape and expose JMX mBeans as Prometheus metrics. For more information, see prometheus/jmx_exporter.

The CloudWatch agent can collect predefined Prometheus metrics from Java Virtual Machine (JVM), Hjava, and Tomcat (Catalina), from a JMX exporter on EC2 instances.

Step 1: Install the CloudWatch agent

The first step is to install the CloudWatch agent on the EC2 instance. For instructions, see Install the CloudWatch agent.

Step 2: Start the Java/JMX workload

The next step is to start the Java/JMX workload.

First, download the latest JMX exporter jar file from the following location: prometheus/jmx_exporter.

Use the jar for your sample application

The example commands in the following sections use `SampleJavaApplication-1.0-SNAPSHOT.jar` as the jar file. Replace these parts of the commands with the jar for your application.

Prepare the JMX exporter configuration

The `config.yaml` file is the JMX exporter configuration file. For more information, see Configuration in the JMX exporter documentation.

Here is a sample configuration for Java and Tomcat.

```
---
lowercaseOutputName: true
lowercaseOutputLabelNames: true

rules:
- pattern: 'java.lang<type=OperatingSystem><>(FreePhysicalMemorySize|
TotalPhysicalMemorySize|FreeSwapSpaceSize|TotalSwapSpaceSize|SystemCpuLoad|
ProcessCpuLoad|OpenFileDescriptorCount|AvailableProcessors)'
  name: java_lang_OperatingSystem_$1
  type: GAUGE

- pattern: 'java.lang<type=Threading><>(TotalStartedThreadCount|ThreadCount)'
  name: java_lang_threading_$1
  type: GAUGE

- pattern: 'Catalina<type=GlobalRequestProcessor, name=\"(\w+-\w+)-(\d+)\"><>(\w+)'
  name: catalina_globalrequestprocessor_$3_total
  labels:
```

```
    port: "$2"
    protocol: "$1"
  help: Catalina global $3
  type: COUNTER

- pattern: 'Catalina<j2eeType=Servlet, WebModule=//([-a-zA-Z0-9+&@#/%?=~_|!:.,;]*[-
a-zA-Z0-9+&@#/%=~_|]), name=([-a-zA-Z0-9+/$%~_-|!.]*), J2EEApplication=none,
 J2EEServer=none><>(requestCount|maxTime|processingTime|errorCount)'
  name: catalina_servlet_$3_total
  labels:
    module: "$1"
    servlet: "$2"
  help: Catalina servlet $3 total
  type: COUNTER

- pattern: 'Catalina<type=ThreadPool, name="(\w+-\w+)-(\d+)"><>(currentThreadCount|
currentThreadsBusy|keepAliveCount|pollerThreadCount|connectionCount)'
  name: catalina_threadpool_$3
  labels:
    port: "$2"
    protocol: "$1"
  help: Catalina threadpool $3
  type: GAUGE

- pattern: 'Catalina<type=Manager, host=([-a-zA-Z0-9+&@#/%?=~_|!:.,;]*[-a-zA-
Z0-9+&@#/%=~_|]), context=([-a-zA-Z0-9+/$%~_-|!.]*)><>(processingTime|sessionCounter|
rejectedSessions|expiredSessions)'
  name: catalina_session_$3_total
  labels:
    context: "$2"
    host: "$1"
  help: Catalina session $3 total
  type: COUNTER

- pattern: ".*"
```

Start the Java application with the Prometheus exporter

Start the sample application. This will emit Prometheus metrics to port 9404. Be sure to replace the entry point `com.gubupt.sample.app.App` with the correct information for your sample java application.

On Linux, enter the following command.

```
$ nohup java -javaagent:./jmx_prometheus_javaagent-0.14.0.jar=9404:./config.yaml -cp
  ./SampleJavaApplication-1.0-SNAPSHOT.jar com.gubupt.sample.app.App &
```

On Windows, enter the following command.

```
PS C:\> java -javaagent:.\jmx_prometheus_javaagent-0.14.0.jar=9404:.\config.yaml -cp  .
\SampleJavaApplication-1.0-SNAPSHOT.jar com.gubupt.sample.app.App
```

Verify the Prometheus metrics emission

Verify that Prometheus metrics are being emitted.

On Linux, enter the following command.

```
$ curl localhost:9404
```

On Windows, enter the following command.

```
PS C:\> curl  http://localhost:9404
```

Sample output on Linux:

```
StatusCode        : 200
StatusDescription : OK
Content           : # HELP jvm_classes_loaded The number of classes that are currently
 loaded in the JVM
                    # TYPE jvm_classes_loaded gauge
                    jvm_classes_loaded 2526.0
                    # HELP jvm_classes_loaded_total The total number of class...
RawContent        : HTTP/1.1 200 OK
                    Content-Length: 71908
                    Content-Type: text/plain; version=0.0.4; charset=utf-8
                    Date: Fri, 18 Dec 2020 16:38:10 GMT

                    # HELP jvm_classes_loaded The number of classes that are
 currentl...
Forms             : {}
Headers           : {[Content-Length, 71908], [Content-Type, text/plain; version=0.0.4;
 charset=utf-8], [Date, Fri, 18
                    Dec 2020 16:38:10 GMT]}
Images            : {}
InputFields       : {}
```

```
Links              : {}
ParsedHtml         : System.__ComObject
RawContentLength   : 71908
```

Step 3: Configure the CloudWatch agent to scrape Prometheus metrics

Next, set up the Prometheus scrape configuration in the CloudWatch agent configuration file.

To set up the Prometheus scrape configuration for the Java/JMX example

1. Set up the configuration for `file_sd_config` and `static_config`.

   On Linux, enter the following command.

```
$ cat /opt/aws/amazon-cloudwatch-agent/var/prometheus.yaml
global:
  scrape_interval: 1m
  scrape_timeout: 10s
scrape_configs:
  - job_name: jmx
    sample_limit: 10000
    file_sd_configs:
      - files: [ "/opt/aws/amazon-cloudwatch-agent/var/prometheus_file_sd.yaml" ]
```

   On Windows, enter the following command.

```
PS C:\ProgramData\Amazon\AmazonCloudWatchAgent> cat prometheus.yaml
global:
  scrape_interval: 1m
  scrape_timeout: 10s
scrape_configs:
  - job_name: jmx
    sample_limit: 10000
    file_sd_configs:
      - files: [ "C:\\ProgramData\\Amazon\\AmazonCloudWatchAgent\
\prometheus_file_sd.yaml" ]
```

2. Set up the scrape targets configuration.

   On Linux, enter the following command.

```
$ cat /opt/aws/amazon-cloudwatch-agent/var/prometheus_file_sd.yaml
```

```
- targets:
  - 127.0.0.1:9404
  labels:
    application: sample_java_app
    os: linux
```

On Windows, enter the following command.

```
PS C:\ProgramData\Amazon\AmazonCloudWatchAgent> cat prometheus_file_sd.yaml
- targets:
  - 127.0.0.1:9404
  labels:
    application: sample_java_app
    os: windows
```

3. Set up the Prometheus scrape configuration by `ec2_sc_config`. Replace *`your-ec2-instance-id`* with the correct EC2 instance ID.

   On Linux, enter the following command.

```
$ cat .\prometheus.yaml
global:
  scrape_interval: 1m
  scrape_timeout: 10s
scrape_configs:
  - job_name: jmx
    sample_limit: 10000
    ec2_sd_configs:
      - region: us-east-1
        port: 9404
        filters:
          - name: instance-id
            values:
              - your-ec2-instance-id
```

   On Windows, enter the following command.

```
PS C:\ProgramData\Amazon\AmazonCloudWatchAgent> cat prometheus_file_sd.yaml
- targets:
  - 127.0.0.1:9404
  labels:
    application: sample_java_app
```

```
    os: windows
```

4. Set up the CloudWatch agent configuration. First, navigate to the correct directory. On Linux, it is `/opt/aws/amazon-cloudwatch-agent/var/cwagent-config.json`. On Windows, it is `C:\ProgramData\Amazon\AmazonCloudWatchAgent\cwagent-config.json`.

   The following is a sample configuration with Java/JHX Prometheus metrics defined. Be sure to replace *`path-to-Prometheus-Scrape-Configuration-file`* with the correct path.

```
{
  "agent": {
    "region": "us-east-1"
  },
  "logs": {
    "metrics_collected": {
      "prometheus": {
        "cluster_name": "my-cluster",
        "log_group_name": "prometheus-test",
        "prometheus_config_path": "path-to-Prometheus-Scrape-Configuration-file",
        "emf_processor": {
          "metric_declaration_dedup": true,
          "metric_namespace": "PrometheusTest",
          "metric_unit":{
            "jvm_threads_current": "Count",
            "jvm_classes_loaded": "Count",
            "java_lang_operatingsystem_freephysicalmemorysize": "Bytes",
            "catalina_manager_activesessions": "Count",
            "jvm_gc_collection_seconds_sum": "Seconds",
            "catalina_globalrequestprocessor_bytesreceived": "Bytes",
            "jvm_memory_bytes_used": "Bytes",
            "jvm_memory_pool_bytes_used": "Bytes"
          },
          "metric_declaration": [
            {
              "source_labels": ["job"],
              "label_matcher": "^jmx$",
              "dimensions": [["instance"]],
              "metric_selectors": [
                "^jvm_threads_current$",
                "^jvm_classes_loaded$",
                "^java_lang_operatingsystem_freephysicalmemorysize$",
                "^catalina_manager_activesessions$",
                "^jvm_gc_collection_seconds_sum$",
```

```
                "^catalina_globalrequestprocessor_bytesreceived$"
              ]
            },
            {
              "source_labels": ["job"],
              "label_matcher": "^jmx$",
              "dimensions": [["area"]],
              "metric_selectors": [
                "^jvm_memory_bytes_used$"
              ]
            },
            {
              "source_labels": ["job"],
              "label_matcher": "^jmx$",
              "dimensions": [["pool"]],
              "metric_selectors": [
                "^jvm_memory_pool_bytes_used$"
              ]
            }
          ]
        }
      }
    },
    "force_flush_interval": 5
  }
}
```

5. Restart the CloudWatch agent by entering one of the following commands.

   On Linux, enter the following command.

```
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-
config -m ec2 -s -c file:/opt/aws/amazon-cloudwatch-agent/var/cwagent-config.json
```

   On Windows, enter the following command.

```
& "C:\Program Files\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1"
 -a fetch-config -m ec2 -s -c file:C:\ProgramData\Amazon\AmazonCloudWatchAgent
\cwagent-config.json
```

Viewing the Prometheus metrics and logs

You can now view the Java/JMX metrics being collected.

To view the metrics for your sample Java/JMX workload

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the Region where your cluster is running, choose **Metrics** in the left navigation pane. Find the **PrometheusTest** namespace to see the metrics.
3. To see the CloudWatch Logs events, choose **Log groups** in the navigation pane. The events are in the log group **prometheus-test**.

Configure CloudWatch agent service and environment names for related entities

The CloudWatch agent can send metrics and logs with entity data to support the Explore related pane in the CloudWatch console. The service name or environment name can be configured by the CloudWatch Agent JSON configuration.

> **Note**
>
> The agent configuration may be overridden. For details about how the agent decides what data to send for related entities, see Using the CloudWatch agent with related telemetry.

For metrics, it can be configured at the agent, metrics, or plugin level. For logs it can be configured at the agent, logs, or file level. The most specific configuration is always used. For example if the configuration exists at the agent level and metrics level, then metrics will use the metric configuration, and anything else (logs) will use the agent configuration. The following example shows different ways to configure the service name and environment name.

```
{
  "agent": {
    "service.name": "agent-level-service",
    "deployment.environment": "agent-level-environment"
  },

  "metrics": {
    "service.name": "metric-level-service",
     "deployment.environment": "metric-level-environment",
```

```
    "metrics_collected": {
      "statsd": {
        "service.name": "statsd-level-service",
        "deployment.environment": "statsd-level-environment",
      },
      "collectd": {
        "service.name": "collectdd-level-service",
        "deployment.environment": "collectd-level-environment",
      }
    }

  },

  "logs": {
    "service.name": "log-level-service",
    "deployment.environment": "log-level-environment",

    "logs_collected": {
      "files": {
        "collect_list": [
          {
            "file_path": "/opt/aws/amazon-cloudwatch-agent/logs/amazon-cloudwatch-
agent.log",
            "log_group_name": "amazon-cloudwatch-agent.log",
            "log_stream_name": "amazon-cloudwatch-agent.log",

            "service.name": "file-level-service",
            "deployment.environment": "file-level-environment"
          }
        ]
      }
    }

  }
}
```

Install the CloudWatch agent with the Amazon CloudWatch Observability EKS add-on or the Helm chart

You can use either the Amazon CloudWatch Observability EKS add-on or the Amazon CloudWatch Observability Helm chart to install the CloudWatch Agent and the Fluent-bit agent on an Amazon

EKS cluster. You can also use the Helm chart to install the CloudWatch Agent and the Fluent-bit agent on a Kubernetes cluster that is not hosted on Amazon EKS.

Using either method on an Amazon EKS cluster enables both Container Insights with enhanced observability for Amazon EKS and CloudWatch Application Signals by default. Both features help you to collect infrastructure metrics, application performance telemetry, and container logs from the cluster.

With Container Insights with enhanced observability for Amazon EKS, Container Insights metrics are charged per observation instead of being charged per metric stored or log ingested. For Application Signals, billing is based on inbound requests to your applications, outbound requests from your applications, and each configured service level objective (SLO). Each inbound request received generates one application signal, and each outbound request made generates one application signal. Every SLO creates two application signals per measurement period. For more information about CloudWatch pricing, see Amazon CloudWatch Pricing.

Both methods enable Container Insights on both Linux and Windows worker nodes in the Amazon EKS cluster. To enable Container Insights on Windows, you must use version 1.5.0 or later of the Amazon EKS add-on or the Helm chart. Currently, Application Signals is not supported on Windows in Amazon EKS clusters.

The Amazon CloudWatch Observability EKS add-on is supported on Amazon EKS clusters running with Kubernetes version 1.23 or later.

When you install the add-on or the Helm chart, you must also grant IAM permissions to enable the CloudWatch agent to send metrics, logs, and traces to CloudWatch. There are two ways to do this:

- Attach a policy to the IAM role of your worker nodes. This option grants permissions to worker nodes to send telemetry to CloudWatch.
- Use an IAM role for service accounts for the agent pods, and attach the policy to this role. This works only for Amazon EKS clusters. This option gives CloudWatch access only to the appropriate agent pods.

Option 1: Install using EKS Pod Identity

If you use version 3.1.0 or later of the add-on, you can use EKS Pod Identity to grant the required permissions to the add-on. EKS Pod Identity is the recommended option and provides benefits such as least privilege, credential rotation, and auditability. Additionally, using EKS Pod Identity allows you to install the EKS add-on as part of the cluster creation itself.

To use this method, first follow the EKS Pod Identity association steps to create the IAM role and set up the EKS Pod Identity agent.

Then install the Amazon CloudWatch Observability EKS add-on. To install the add-on, you can use the Amazon CLI, the Amazon EKS console, Amazon CloudFormation, or Terraform.

Amazon CLI

To use the Amazon CLI to install the Amazon CloudWatch Observability EKS add-on

Enter the following commands. Replace *my-cluster-name* with the name of your cluster and replace *111122223333* with your account ID. Replace *my-role* with the IAM role that you created in the EKS Pod Identity association step.

```
aws iam attach-role-policy \
--role-name my-role \
--policy-arn arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy

aws eks create-addon \
--addon-name amazon-cloudwatch-observability \
--cluster-name my-cluster-name \
--pod-identity-associations serviceAccount=cloudwatch-
agent,roleArn=arn:aws:iam::111122223333:role/my-role
```

Amazon EKS console

To use the Amazon EKS console to add the Amazon CloudWatch Observability EKS add-on

1. Open the Amazon EKS console at https://console.amazonaws.cn/eks/home#/clusters.
2. In the left navigation pane, choose **Clusters**.
3. Choose the name of the cluster that you want to configure the Amazon CloudWatch Observability EKS add-on for.
4. Choose the **Add-ons** tab.
5. Choose **Get more add-ons**.
6. On the **Select add-ons** page, do the following:
    a. In the **Amazon EKS-addons** section, select the **Amazon CloudWatch Observability** check box.
    b. Choose **Next**.

7. On the **Configure selected add-ons settings** page, do the following:

   a. Select the **Version** you'd like to use.

   b. For **Add-on access**, select **EKS Pod Identity**

   c. If you don't have an IAM role configured, choose **Create recommended role**, then choose **Next** until you are at **Step 3 Name, review, and create**. You can change your role name if desired, otherwise, choose **Create Role**, and then return to the Add-on page and select the IAM role that you just created.

   d. (Optional) You can expand the **Optional configuration settings**. If you select **Override** for the **Conflict resolution method**, one or more of the settings for the existing add-on can be overwritten with the Amazon EKS add-on settings. If you don't enable this option and there's a conflict with your existing settings, the operation fails. You can use the resulting error message to troubleshoot the conflict. Before selecting this option, make sure that the Amazon EKS add-on doesn't manage settings that you need to self-manage.

   e. Choose **Next**.

8. On the **Review and add** page, choose **Create**. After the add-on installation is complete, you see your installed add-on.

Amazon CloudFormation

To use Amazon CloudFormation to install the Amazon CloudWatch Observability EKS add-on

1. First, run the following Amazon CLI command to attach the necessary IAM policy to your IAM role. Replace *my-role* with the role that you created in the EKS Pod Identity association step.

```
aws iam attach-role-policy \
--role-name my-role \
--policy-arn arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy
```

2. Then create the following resource. Replace *my-cluster-name* with the name of your cluster, replace *111122223333* with your account ID, and replace *my-role* with the IAM role created in EKS Pod Identity association step. For more information, see AWS::EKS::Addon.

```
{
    "Resources": {
        "EKSAddOn": {
            "Type": "AWS::EKS::Addon",
            "Properties": {
                "AddonName": "amazon-cloudwatch-observability",
                "ClusterName": "my-cluster-name",
                "PodIdentityAssociations": [
                    {
                        "ServiceAccount": "cloudwatch-agent",
                        "RoleArn": "arn:aws:iam::111122223333:role/my-role"
                    }
                ]
            }
        }
    }
}
```

Terraform

To use Terraform to install the Amazon CloudWatch Observability EKS add-on

1. Use the following. Replace *my-cluster-name* with the name of your cluster, replace *111122223333* with your account ID, and replace *my-service-account-role* with the IAM role created in EKS Pod Identity association step.

   For more information, see [Resource: aws_eks_addon](#) in the Terraform documentation.

2.
```
resource "aws_iam_role_policy_attachment" "CloudWatchAgentServerPolicy" {
  policy_arn = "arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy"
  role       = "my-role"
}

resource "aws_eks_addon" "example" {
  cluster_name = "my-cluster-name"
  addon_name   = "amazon-cloudwatch-observability"
  pod_identity_associations {
      roleArn = "arn:aws:iam::111122223333:role/my-role"
      serviceAccount = "cloudwatch-agent"
  }
}
```

Option 2: Install with IAM permissions on worker nodes

To use this method, first attach the **CloudWatchAgentServerPolicy** IAM policy to your worker nodes by entering the following command. In this command, replace *my-worker-node-role* with the IAM role used by your Kubernetes worker nodes.

```
aws iam attach-role-policy \
--role-name my-worker-node-role \
--policy-arn arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy
```

Then install the Amazon CloudWatch Observability EKS add-on. To install the add-on, you can use the Amazon CLI, the console, Amazon CloudFormation, or Terraform.

Amazon CLI

To use the Amazon CLI to install the Amazon CloudWatch Observability EKS add-on

Enter the following command. Replace *my-cluster-name* with the name of your cluster.

```
aws eks create-addon --addon-name amazon-cloudwatch-observability --cluster-name my-cluster-name
```

Amazon EKS console

To use the Amazon EKS console to add the Amazon CloudWatch Observability EKS add-on

1. Open the Amazon EKS console at https://console.amazonaws.cn/eks/home#/clusters.
2. In the left navigation pane, choose **Clusters**.
3. Choose the name of the cluster that you want to configure the Amazon CloudWatch Observability EKS add-on for.
4. Choose the **Add-ons** tab.
5. Choose **Get more add-ons**.
6. On the **Select add-ons** page, do the following:
   a. In the **Amazon EKS-addons** section, select the **Amazon CloudWatch Observability** check box.
   b. Choose **Next**.
7. On the **Configure selected add-ons settings** page, do the following:

   a. Select the **Version** you'd like to use.
   b. (Optional) You can expand the **Optional configuration settings**. If you select **Override** for the **Conflict resolution method**, one or more of the settings for the existing add-on can be overwritten with the Amazon EKS add-on settings. If you don't enable this option and there's a conflict with your existing settings, the operation fails. You can use the resulting error message to troubleshoot the conflict. Before selecting this option, make sure that the Amazon EKS add-on doesn't manage settings that you need to self-manage.
   c. Choose **Next**.

8. On the **Review and add** page, choose **Create**. After the add-on installation is complete, you see your installed add-on.

Amazon CloudFormation

To use Amazon CloudFormation to install the Amazon CloudWatch Observability EKS add-on

Replace *my-cluster-name* with the name of your cluster. For more information, see AWS::EKS::Addon.

```
{
    "Resources": {
        "EKSAddOn": {
            "Type": "AWS::EKS::Addon",
            "Properties": {
                "AddonName": "amazon-cloudwatch-observability",
                "ClusterName": "my-cluster-name"
            }
        }
    }
}
```

Helm chart

To use the `amazon-cloudwatch-observability` Helm chart

1. You must have Helm installed to use this chart. For more information about installing Helm, see the Helm documentation.

2. After you have installed Helm, enter the following commands. Replace *my-cluster-name* with the name of your cluster, and replace *my-cluster-region* with the Region that the cluster runs in.

```
helm repo add aws-observability https://aws-observability.github.io/helm-charts
helm repo update aws-observability
helm install --wait --create-namespace --namespace amazon-cloudwatch amazon-
cloudwatch-observability aws-observability/amazon-cloudwatch-observability --set
 clusterName=my-cluster-name --set region=my-cluster-region
```

Terraform

To use Terraform to install the Amazon CloudWatch Observability EKS add-on

Replace *my-cluster-name* with the name of your cluster. For more information, see [Resource: aws_eks_addon](#).

```
resource "aws_eks_addon" "example" {
  addon_name   = "amazon-cloudwatch-observability"
  cluster_name = "my-cluster-name"
}
```

Option 3: Install using IAM service account role (applies only to using the add-on)

This method is valid only if you are using the Amazon CloudWatch Observability EKS add-on. Before using this method, verify the following prerequisites:

- You have a functional Amazon EKS cluster with nodes attached in one of the Amazon Web Services Regions that supports Container Insights. For the list of supported Regions, see Container Insights.
- You have `kubectl` installed and configured for the cluster. For more information, see Installing `kubectl` in the *Amazon EKS User Guide*.
- You have `eksctl` installed. For more information, see Installing or updating `eksctl` in the *Amazon EKS User Guide*.

Amazon CLI

To use the Amazon CLI to install the Amazon CloudWatch Observability EKS add-on using the IAM service account role

1. Enter the following command to create an OpenID Connect (OIDC) provider, if the cluster doesn't have one already. For more information, see Configuring a Kubernetes service account to assume an IAM role in the *Amazon EKS User Guide*.

```
eksctl utils associate-iam-oidc-provider --cluster my-cluster-name --approve
```

2. Enter the following command to create the IAM role with the **CloudWatchAgentServerPolicy** policy attached, and configure the agent service account to assume that role using OIDC. Replace *my-cluster-name* with the name of your cluster, and replace *my-service-account-role* with the name of the role that you want to associate the service account with. If the role doesn't already exist, `eksctl` creates it for you.

```
eksctl create iamserviceaccount \
  --name cloudwatch-agent \
  --namespace amazon-cloudwatch --cluster my-cluster-name \
  --role-name my-service-account-role \
  --attach-policy-arn arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy \
  --role-only \
  --approve
```

3. Install the add-on by entering the following command. Replace *my-cluster-name* with the name of your cluster, replace *111122223333* with your account ID, and replace *my-service-account-role* with the IAM role created in the previous step.

```
aws eks create-addon --addon-name amazon-cloudwatch-observability
 --cluster-name my-cluster-name --service-account-role-arn
 arn:aws:iam::111122223333:role/my-service-account-role
```

Amazon EKS console

To use the console to install the Amazon CloudWatch Observability EKS add-on using the IAM service account role

1. Open the Amazon EKS console at https://console.amazonaws.cn/eks/home#/clusters.
2. In the left navigation pane, choose **Clusters**.
3. Choose the name of the cluster that you want to configure the Amazon CloudWatch Observability EKS add-on for.
4. Choose the **Add-ons** tab.
5. Choose **Get more add-ons**.
6. On the **Select add-ons** page, do the following:

   a. In the **Amazon EKS-addons** section, select the **Amazon CloudWatch Observability** check box.

   b. Choose **Next**.
7. On the **Configure selected add-ons settings** page, do the following:

   a. Select the **Version** you'd like to use.

   b. For **Add-on access**, select **IAM roles for service accounts (IRSA)**

   c. Select the IAM role in the **Add-on access** box.

   d. (Optional) You can expand the **Optional configuration settings**. If you select **Override** for the **Conflict resolution method**, one or more of the settings for the existing add-on can be overwritten with the Amazon EKS add-on settings. If you don't enable this option and there's a conflict with your existing settings, the operation fails. You can use the resulting error message to troubleshoot the conflict. Before selecting this option, make sure that the Amazon EKS add-on doesn't manage settings that you need to self-manage.

   e. Choose **Next**.
8. On the **Review and add** page, choose **Create**. After the add-on installation is complete, you see your installed add-on.

Considerations for Amazon EKS Hybrid Nodes

Node-level metrics aren't available for hybrid nodes because Container Insights depends on the availability of the EC2 Instance Metadata Service (IMDS) for node-level metrics. Cluster, workload, Pod, and container-level metrics are available for hybrid nodes.

After you install the add-on by following the steps in the previous sections, you must update the add-on manifest so that the agent can run successfully on hybrid nodes. Edit the `amazoncloudwatchagents` resource in the cluster to add the `RUN_WITH_IRSA` environment variable to match the following.

```
kubectl edit amazoncloudwatchagents -n amazon-cloudwatch cloudwatch-agent
```

```
apiVersion: v1
       items:
       - apiVersion: cloudwatch.aws.amazon.com/v1alpha1
         kind: AmazonCloudWatchAgent
         metadata:
           ...
           name: cloudwatch-agent
           namespace: amazon-cloudwatch
           ...
         spec:
           ...
           env:
           - name: RUN_WITH_IRSA # <-- Add this
             value: "True" # <-- Add this
           - name: K8S_NODE_NAME
             valueFrom:
               fieldRef:
                 fieldPath: spec.nodeName
                 ...
```

(Optional) Additional configuration

Topics

- Opt out of collecting container logs
- Use a custom Fluent Bit configuration
- Manage Kubernetes tolerations for the installed pod workloads

- Opt out of accelerated compute metrics collection
- Use a custom CloudWatch agent configuration
- Manage admission webhook TLS certificates
- Collect Amazon EBS volume IDs

Opt out of collecting container logs

By default, the add-on uses Fluent Bit to collect container logs from all pods and then sends the logs to CloudWatch Logs. For information about which logs are collected, see Setting up Fluent Bit.

> **Note**
>
> Neither the add-on or the Helm chart manage existing Fluentd or Fluent Bit resources in a cluster. You can delete the existing Fluentd or Fluent Bit resources before installing the add-on or Helm chart. Alternatively, to keep your existing setup and avoid having the add-on or the Helm chart from also installing Fluent Bit, you can disable it by following the instructions in this section.

To opt out of the collection of container logs if you are using the Amazon CloudWatch Observability EKS add-on, pass the following option when you create or update the add-on:

```
--configuration-values '{ "containerLogs": { "enabled": false } }'
```

To opt out of the collection of container logs if you are using the Helm chart, pass the following option when you create or update the add-on:

```
--set containerLogs.enabled=false
```

Use a custom Fluent Bit configuration

Starting with version 1.7.0 of the Amazon CloudWatch Observability EKS add-on, you can modify the Fluent Bit configuration when you create or update the add-on or Helm chart. You supply the custom Fluent Bit configuration in the `containerLogs` root level section of the advanced configuration of the add-on or the value overrides in the Helm chart. Within this section, you supply the custom Fluent Bit configuration in the `config` section (for Linux) or `configWindows` section (for Windows). The `config` is further broken down into the following sub-sections:

- `service`– This section represents the `SERVICE` config to define the global behavior of the Fluent Bit engine.
- `customParsers`– This section represents any global `PARSER`s that you want to include that are capable of taking unstructured log entries and giving them a structure to make it easier for processing and further filtering.
- `extraFiles`– This section can be used to provide additional Fluent Bit `conf` files to be included. By default, the following 3 `conf` files are included:.
  - `application-log.conf`– A `conf` file for sending application logs from your cluster to the log group `/aws/containerinsights/my-cluster-name/application` in CloudWatch Logs.
  - `dataplane-log.conf`– A `conf` file for sending logs corresponding to your cluster's data plane components including the CRI logs, kubelet logs, kube-proxy logs and Amazon VPC CNI logs to the log group `/aws/containerinsights/my-cluster-name/dataplane` in CloudWatch Logs.
  - `host-log.conf`– A `conf` for sending logs from `/var/log/dmesg`, `/var/log/messages`, and `/var/log/secure` on Linux, and System `winlogs` on Windows, to the log group `/aws/containerinsights/my-cluster-name/host` in CloudWatch.

Note

Provide the full configuration for each of these individual sections even if you are modifying only one field within a sub-section. We recommend that you use the default configuration provided below as a baseline and then modify it accordingly so that you don't disable functionality that is enabled by default. You can use the following YAML configuration when modifying the advanced config for the Amazon EKS add-on or when you supply value overrides for the Helm chart.

To find the `config` section for your cluster, see aws-observability / helm-charts on GitHub and find the release corresponding to the version of the add-on or Helm chart that you are installing. Then navigate to `/charts/amazon-cloudwatch-observability/values.yaml` to find the `config` section (for Linux) and `configWindows` section (for Windows) within the `fluentBit` section under `containerLogs`.

As an example, the default Fluent Bit configuration for version 1.7.0 can be found here.

We recommend that you provide the `config` as YAML when you supply it using the Amazon EKS add-on's advanced config or when you supply it as value overrides for your Helm installation. Be sure that the YAML conforms to the following structure.

```
containerLogs:
  fluentBit:
    config:
      service: |
        ...
      customParsers: |
        ...
      extraFiles:
        application-log.conf: |
          ...
        dataplane-log.conf: |
          ...
        host-log.conf: |
          ...
```

The following example `config` changes the global setting for the flush interval to be 45 seconds. Even though the only modification is to the `Flush` field, you must still provide the full `SERVICE` definition for the service sub-section. Because this example didn't specify overrides for the other sub-sections, the defaults are used for them.

```
containerLogs:
  fluentBit:
    config:
      service: |
        [SERVICE]
          Flush                     45
          Grace                     30
          Log_Level                 error
          Daemon                    off
          Parsers_File              parsers.conf
          storage.path              /var/fluent-bit/state/flb-storage/
          storage.sync              normal
          storage.checksum          off
          storage.backlog.mem_limit 5M
```

The following example configuration includes an extra Fluent bit `conf` file. In this example, we are adding a custom `my-service.conf` under `extraFiles` and it will be included in addition to the three default `extraFiles`.

```
containerLogs:
  fluentBit:
    config:
      extraFiles:
        my-service.conf: |
          [INPUT]
            Name                tail
            Tag                 myservice.*
            Path                /var/log/containers/*myservice*.log
            DB                  /var/fluent-bit/state/flb_myservice.db
            Mem_Buf_Limit       5MB
            Skip_Long_Lines     On
            Ignore_Older        1d
            Refresh_Interval    10

          [OUTPUT]
            Name                  cloudwatch_logs
            Match                 myservice.*
            region                ${AWS_REGION}
            log_group_name        /aws/containerinsights/${CLUSTER_NAME}/myservice
            log_stream_prefix     ${HOST_NAME}-
            auto_create_group     true
```

The next example removes an existing `conf` file entirely from `extraFiles`. This excludes the `application-log.conf` entirely by overriding it with an empty string. Simply omitting `application-log.conf` from `extraFiles` would instead imply to use the default, which is not what we are trying to achieve in this example. The same applies to removing any custom `conf` file that you might have previously added to `extraFiles`.

```
containerLogs:
  fluentBit:
    config:
      extraFiles:
        application-log.conf: ""
```

Manage Kubernetes tolerations for the installed pod workloads

Starting with version 1.7.0 of the Amazon CloudWatch Observability EKS add-on, the add-on and the Helm chart by default set Kubernetes *tolerations* to tolerate all taints on the pod workloads that are installed by the add-on or the Helm chart. This ensures that daemonsets such as the CloudWatch agent and Fluent Bit can schedule pods on all nodes in your cluster by default. For more information about tolerations and taints, see Taints and Tolerations in the Kubernetes documentation.

The default tolerations set by the add-on or the Helm chart are as follows:

```
tolerations:
- operator: Exists
```

You can override the default tolerations by setting the `tolerations` field at the root level when using the add-on advanced config or when you install or upgrade the Helm chart with value overrides. An example would look like the following:

```
tolerations:
- key: "key1"
  operator: "Exists"
  effect: "NoSchedule"
```

To omit tolerations completely, you can use a config that looks like the following:

```
tolerations: []
```

Any changes to tolerations apply to all pod workloads that are installed by the add-on or the Helm chart.

Opt out of accelerated compute metrics collection

By default, Container Insights with enhanced observability collects metrics for Accelerated Compute monitoring, including NVIDIA GPU metrics, Amazon Neuron metrics for Amazon Trainium and Amazon Inferentia, and Amazon Elastic Fabric Adapter (EFA) metrics.

NVIDIA GPU metrics from Amazon EKS workloads are collected by default beginning with version `v1.3.0-eksbuild.1` of the EKS add-on or the Helm chart and version `1.300034.0` of the CloudWatch agent. For a list of metrics collected and prerequisites, see NVIDIA GPU metrics.

Amazon Neuron metrics for Amazon Trainium and Amazon Inferentia accelerators are collected by default beginning with version `v1.5.0-eksbuild.1` of the EKS add-on or the Helm chart, and version `1.300036.0` of the CloudWatch agent. For a list of metrics collected and prerequisites, see Amazon Neuron metrics for Amazon Trainium and Amazon Inferentia .

Amazon Elastic Fabric Adapter (EFA) metrics from Linux nodes on Amazon EKS clusters are collected by default beginning with version `v1.5.2-eksbuild.1` of the EKS add-on or the Helm chart and version `1.300037.0` of the CloudWatch agent. For a list of metrics collected and prerequisites, see Amazon Elastic Fabric Adapter (EFA) metrics .

You can opt out of collecting these metrics by setting the `accelerated_compute_metrics` field in the CloudWatch agent configuration file to `false`. This field is in the `kubernetes` section of the `metrics_collected` section in the CloudWatch configuration file. The following is an example of an opt-out configuration. For more information about how to use custom CloudWatch agent configurations, see the following section, Use a custom CloudWatch agent configuration.

```
{
  "logs": {
    "metrics_collected": {
      "kubernetes": {
        "enhanced_container_insights": true,
        "accelerated_compute_metrics": false
      }
    }
  }
}
```

Use a custom CloudWatch agent configuration

To collect other metrics, logs or traces using the CloudWatch agent, you can specify a custom configuration while also keeping Container Insights and CloudWatch Application Signals enabled. To do so, embed the CloudWatch agent configuration file within the config key under the agent key of the advanced configuration that you can use when creating or updating the EKS add-on or the Helm chart. The following represents the default agent configuration when you do not provide any additional configuration.

> **Important**
>
> Any custom configuration that you provide using additional configuration settings overrides the default configuration used by the agent. Be cautious not to unintentionally

disable functionality that is enabled by default, such as Container Insights with enhanced observability and CloudWatch Application Signals. In the scenario that you are required to provide a custom agent configuration, we recommend using the following default configuration as a baseline and then modifying it accordingly.

- For using the Amazon CloudWatch observability EKS add-on

```
--configuration-values '{
  "agent": {
    "config": {
      "logs": {
        "metrics_collected": {
          "application_signals": {},
          "kubernetes": {
            "enhanced_container_insights": true
          }
        }
      },
      "traces": {
        "traces_collected": {
          "application_signals": {}
        }
      }
    }

}'
```

- For using the Helm chart

```
--set agent.config='{
  "logs": {
    "metrics_collected": {
      "application_signals": {},
      "kubernetes": {
        "enhanced_container_insights": true
      }
    }
  },
  "traces": {
    "traces_collected": {
      "application_signals": {}
```

```
    }
  }
}'
```

The following example shows the default agent configuration for the CloudWatch agent on Windows. The CloudWatch agent on Windows does not support custom configuration.

```
{
  "logs": {
    "metrics_collected": {
      "kubernetes": {
        "enhanced_container_insights": true
      },
    }
  }
}
```

Manage admission webhook TLS certificates

The Amazon CloudWatch Observability EKS add-on and the Helm chart leverage Kubernetes admission webhooks to validate and mutate `AmazonCloudWatchAgent` and `Instrumentation` custom resource (CR) requests, and optionally Kubernetes pod requests on the cluster if CloudWatch Application Signals is enabled. In Kubernetes, webhooks require a TLS certificate that the API server is configured to trust in order to ensure secure communication.

By default, the Amazon CloudWatch Observability EKS add-on and the Helm chart auto-generate a self-signed CA and a TLS certificate signed by this CA for securing the communication between the API server and the webhook server. This auto-generated certificate has a default expiry of 10 years and is not auto-renewed upon expiry. In addition, the CA bundle and the certificate are re-generated every time the add-on or Helm chart is upgraded or re-installed, thus resetting the expiry. If you want to change the default expiry of the auto-generated certificate, you can use the following additional configurations when creating or updating the add-on. Replace *`expiry-in-days`* with your desired expiry duration in days.

- Use this for the Amazon CloudWatch Observability EKS add-on

```
--configuration-values '{ "admissionWebhooks": { "autoGenerateCert":
 { "expiryDays": expiry-in-days } } }'
```

- Use this for the Helm chart

```
--set admissionWebhooks.autoGenerateCert.expiryDays=expiry-in-days
```

For a more secure and feature-rich certificate authority solution, the add-on has opt-in support for cert-manager, a widely-adopted solution for TLS certificate management in Kubernetes that simplifies the process of obtaining, renewing, managing and using those certificates. It ensures that certificates are valid and up to date, and attempts to renew certificates at a configured time before expiry. cert-manager also facilitates issuing certificates from a variety of supported sources, including Amazon Certificate Manager Private Certificate Authority.

We recommend that you review best practices for management of TLS certificates on your clusters and advise you to opt in to cert-manager for production environments. Note that if you opt-in to enabling cert-manager for managing the admission webhook TLS certificates, you are required to pre-install cert-manager on your Amazon EKS cluster before you install the Amazon CloudWatch Observability EKS add-on or the Helm chart. For more information about available installation options, see cert-manager documentation. After you install it, you can opt in to using cert-manager for managing the admission webhook TLS certificates using the following additional configuration.

- If you are using the Amazon CloudWatch Observability EKS add-on

  ```
  --configuration-values '{ "admissionWebhooks": { "certManager": { "enabled":
   true } } }'
  ```

- If you are using the Helm chart

  ```
  --set admissionWebhooks.certManager.enabled=true
  ```

```
--configuration-values '{ "admissionWebhooks": { "certManager": { "enabled":
 true } } }'
```

The advanced configuration discussed in this section will by default use a SelfSigned issuer.

Collect Amazon EBS volume IDs

If you want to collect Amazon EBS volume IDs in the performance logs, you must add another policy to the IAM role that is attached to the worker nodes or to the service account. Add the following as an inline policy. For more information, see [Adding and Removing IAM Identity Permissions](#).

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "ec2:DescribeVolumes"
            ],
            "Resource": "*",
            "Effect": "Allow"
        }
    ]
}
```

Troubleshooting the Amazon CloudWatch Observability EKS add-on or the Helm chart

Use the following information to help troubleshoot problems with the Amazon CloudWatch Observability EKS add-on or the Helm chart

Topics

Updating and deleting the Amazon CloudWatch Observability EKS add-on or the Helm chart

For instructions about updating or deleting the Amazon CloudWatch Observability EKS add-on, see Managing Amazon EKS add-ons. Use `amazon-cloudwatch-observability` as the name of the add-on.

To delete the Helm chart in a cluster, enter the following command.

```
helm delete amazon-cloudwatch-observability -n amazon-cloudwatch --wait
```

Verify the version of the CloudWatch agent used by the Amazon CloudWatch Observability EKS add-on or the Helm chart

The Amazon CloudWatch Observability EKS add-on and the Helm chart installs a custom resource of kind `AmazonCloudWatchAgent` that controls the behavior of the CloudWatch agent daemonset on the cluster, including the version of the CloudWatch agent being used. You can get a list of all the `AmazonCloudWatchAgent` custom resources installed on your cluster u by entering the following command:

```
kubectl get amazoncloudwatchagent -A
```

In the output of this command, you should be able to check the version of the CloudWatch agent. Alternatively, you can also describe the `amazoncloudwatchagent` resource or one of the `cloudwatch-agent-*` pods running on your cluster to inspect the image being used.

Handling a ConfigurationConflict when managing the add-on or the Helm chart

When you install or update the Amazon CloudWatch Observability EKS add-on or the Helm chart, if you notice a failure caused by existing resources, it is likely because you already have the CloudWatch agent and its associated components such as the ServiceAccount, the ClusterRole and the ClusterRoleBinding installed on the cluster.

The error displayed by the add-on will include `Conflicts found when trying to apply. Will not continue due to resolve conflicts mode`,

The error displayed by the Helm chart will be similar to `Error: INSTALLATION FAILED: Unable to continue with install and invalid ownership metadata.`.

When the add-on or the Helm chart tries to install the CloudWatch agent and its associated components, if it detects any change in the contents, it by default fails the installation or update to avoid overwriting the state of the resources on the cluster.

If you are trying to onboard to the Amazon CloudWatch Observability EKS add-on and you see this failure, we recommend deleting an existing CloudWatch agent setup that you had previously installed on the cluster and then installing the EKS add-on or Helm chart. Be sure to back up any customizations you might have made to the original CloudWatch agent setup such as a custom agent configuration, and provide these to the add-on or Helm chart when you next install or update it. If you had previously installed the CloudWatch agent for onboarding to Container Insights, see Deleting the CloudWatch agent and Fluent Bit for Container Insights for more information.

Alternatively, the add-on supports a conflict resolution configuration option that has the capability to specify `OVERWRITE`. You can use this option to proceed with installing or updating the add-on by overwriting the conflicts on the cluster. If you are using the Amazon EKS console, you'll find the **Conflict resolution method** when you choose the **Optional configuration settings** when you create or update the add-on. If you are using the Amazon CLI, you can supply the `--resolve-conflicts OVERWRITE` to your command to create or update the add-on.

Collect Java Management Extensions (JMX) metrics

The CloudWatch agent supports Java Management Extensions (JMX) metrics collection on Amazon EKS. This allows you to collect additional metrics from Java applications running on Amazon EKS clusters enabling insight into performance, memory usage, traffic, and other critical metrics. For more information, see Collect Java Management Extensions (JMX) metrics.

Enable Kueue metrics

Beginning with version `v2.4.0-eksbuild.1` of the the CloudWatch Observability EKS add-on, Container Insights for Amazon EKS supports collecting Kueue metrics from Amazon EKS clusters. For more information about these metrics, see Kueue metrics .

If you are using the Amazon SageMaker AI Hyperpod Task Governance EKS add-on, you can skip the steps in the **Prerequisites** section and just follow the steps in Enable the configuration flag.

Prerequisites

Before you install Kueue in your Amazon EKS cluster, make the following updates in the manifest file:

1. Enable the optional cluster queue resource metrics for Kueue. To do this, modify the in-line `controller_manager_config.yaml` in the `kueue-system` ConfigMap. In the `metrics` section, add or uncomment the line `enableClusterQueueResources: true`.

```
apiVersion: v1
data:
  controller_manager_config.yaml: |
    apiVersion: config.kueue.x-k8s.io/v1beta1
```

```
    kind: Configuration
    health:
      healthProbeBindAddress: :8081
    metrics:
      bindAddress: :8080
      enableClusterQueueResources: true  <-- ADD/UNCOMMENT THIS LINE
```

2. By default, all `k8s` services are available cluster-wide. Kueue creates a service `kueue-controller-manager-metrics-service` for exposing metrics. To prevent duplicate observations for metrics, modify this service to allow access only to the metrics service from the same node. To do this, add the line `internalTrafficPolicy: Local` to the `kueue-controller-manager-metrics-service` definition.

```
apiVersion: v1
kind: Service
metadata:
  labels:
    ...
  name: kueue-controller-manager-metrics-service
  namespace: kueue-system
spec:
  ports:
  - name: https
    port: 8443
    protocol: TCP
    targetPort: https
  internalTrafficPolicy: Local   <-- ADD THIS LINE
  selector:
    control-plane: controller-manager
```

3. Lastly, the `kueue-controller-manager` pod creates a `kube-rbac-proxy` container. This container currently has a high level of logging verbosity, which causes the cluster's bearer token to be logged by that container when the metrics scraper accesses the `kueue-controller-manager-metrics-service`. We recommend that you decrease this logging verbosity. The default value in the manifest distributed by Kueue is 10, we recommend to change it to 0.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    ...
  name: kueue-controller-manager
```

```
  namespace: kueue-system
spec:
  ...
  template:
    ...
    spec:
      containers:
      ...
      - args:
        - --secure-listen-address=0.0.0.0:8443
        - --upstream=http://127.0.0.1:8080/
        - --logtostderr=true
        - --v=0  <-- CHANGE v=10 TO v=0
        image: gcr.io/kubebuilder/kube-rbac-proxy:v0.8.0
        name: kube-rbac-proxy
        ...
```

Enable the configuration flag

To enable the Kueue metrics, you must enable `kueue_container_insights` in the add-on additional configuration. You can do this either by using the Amazon CLI to set up the EKS Observability add-on, or by using the Amazon EKS console.

After you have successfully installed the EKS Observability add-on with one of the following methods, you can view your Amazon EKS cluster metrics under the HyperPod console **Dashboard** tab.

Amazon CLI

To enable Kueue metrics using the Amazon CLI

- Enter the following Amazon CLI command to install the add-on.

  ```
  aws eks create-addon --cluster-name cluster-name --addon-name amazon-cloudwatch-observability --configuration-values "configuration_json_file"
  ```

  The following is an example of the JSON file with the configuration values.

  ```
  {
      "agent": {
          "config": {
  ```

```
            "logs": {
                "metrics_collected": {
                    "kubernetes": {
                        "kueue_container_insights": true,
                        "enhanced_container_insights": true
                    },
                    "application_signals": { }
                }
            },
            "traces": {
                "traces_collected": {
                    "application_signals": { }
                }
            }
        },
    },
}
```

Amazon EKS console

To enable Kueue metrics using the Amazon EKS console

1. Open the Amazon EKS console at https://console.amazonaws.cn/eks/home#/clusters.
2. Choose the name of your cluster.
3. Choose **Add-ons**.
4. Find the **Amazon CloudWatch Observability** add-on in the list, and install it. When you do so, choose **Optional configuration** and include the following JSON configuration values.

```
{
    "agent": {
        "config": {
            "logs": {
                "metrics_collected": {
                    "kubernetes": {
                        "kueue_container_insights": true,
                        "enhanced_container_insights": true
                    },
                    "application_signals": { }
                }
            },
            "traces": {
```

```
                "traces_collected": {
                    "application_signals": { }
                }
            }
        },
    },
}
```

Appending OpenTelemetry collector configuration files

The CloudWatch agent supports supplemental OpenTelemetry collector configuration files alongside its own configuration files. This feature allows you to use CloudWatch agent features such as CloudWatch Application Signals or Container Insights through the CloudWatch agent configuration and bring in your existing OpenTelemetry collector configuration with a single agent.

To prevent merge conflicts with pipelines automatically created by CloudWatch agent, we recommend that you add a custom suffix to each of the components and pipelines in your OpenTelemetry collector configuration. This will prevent clashing and merge conflicts.

- If you are using the Amazon CloudWatch Observability EKS add-on

  ```
  --configuration-values file://values.yaml
  ```

  or

  ```
  --configuration-values '
    agent:
      otelConfig:
        receivers:
          otlp/custom-suffix:
            protocols:
              http: {}
        exporters:
          awscloudwatchlogs/custom-suffix:
            log_group_name: "test-group"
            log_stream_name: "test-stream"
        service:
          pipelines:
            logs/custom-suffix:
              receivers: [otlp/custom-suffix]
  ```

```
            exporters: [awscloudwatchlogs/custom-suffix]
'
```

- If you are using the Helm chart

```
--set agent.otelConfig='
  receivers:
    otlp/custom-suffix:
      protocols:
        http: {}
  exporters:
    awscloudwatchlogs/custom-suffix:
      log_group_name: "test-group"
      log_stream_name: "test-stream"
  service:
    pipelines:
      logs/custom-suffix:
        receivers: [otlp/custom-suffix]
        exporters: [awscloudwatchlogs/custom-suffix]
'
```

Metrics collected by the CloudWatch agent

You can collect metrics from servers by installing the CloudWatch agent on the server. You can install the agent on both Amazon EC2 instances and on-premises servers. You can also install the agent on computers running Linux, Windows Server, or macOS. If you install the agent on an Amazon EC2 instance, the metrics the agent collects are in addition to the metrics enabled by default on Amazon EC2 instances. For information about installing the CloudWatch agent on an instance, see Collect metrics, logs, and traces with the CloudWatch agent. You can use this section to learn about metrics the CloudWatch agent collects.

Metrics collected by the CloudWatch agent on Windows Server instances

On a server running Windows Server, installing the CloudWatch agent enables you to collect the metrics associated with the counters in Windows Performance Monitor. The CloudWatch metric names for these counters are created by putting a space between the object name and the counter name. For example, the `% Interrupt Time` counter of the `Processor` object is given the metric name `Processor % Interrupt Time` in CloudWatch. For more information about Windows Performance Monitor counters, see the Microsoft Windows Server documentation.

The default namespace for metrics collected by the CloudWatch agent is `CWAgent`, although you can specify a different namespace when you configure the agent.

Metrics collected by the CloudWatch agent on Linux and macOS instances

The following table lists the metrics that you can collect with the CloudWatch agent on Linux servers and macOS computers.

Metric	Description
`cpu_time_active`	The amount of time that the CPU is active in any capacity. This metric is measured in hundredths of a second. Unit: None
`cpu_time_guest`	The amount of time that the CPU is running a virtual CPU for a guest operating system. This metric is measured in hundredths of a second. Unit: None
`cpu_time_guest_nice`	The amount of time that the CPU is running a virtual CPU for a guest operating system, which is low-priority and can be interrupted by other processes. This metric is measured in hundredths of a second. Unit: None
`cpu_time_idle`	The amount of time that the CPU is idle. This metric is measured in hundredths of a second. Unit: None

Metric	Description
`cpu_time_iowait`	The amount of time that the CPU is waiting for I/O operations to complete. This metric is measured in hundredths of a second. Unit: None
`cpu_time_irq`	The amount of time that the CPU is servicing interrupts. This metric is measured in hundredths of a second. Unit: None
`cpu_time_nice`	The amount of time that the CPU is in user mode with low-priority processes, which can easily be interrupted by higher-priority processes. This metric is measured in hundredths of a second. Unit: None
`cpu_time_softirq`	The amount of time that the CPU is servicing software interrupts. This metric is measured in hundredths of a second. Unit: None
`cpu_time_steal`	The amount of time that the CPU is in *stolen time*, which is time spent in other operating systems in a virtualized environment. This metric is measured in hundredths of a second. Unit: None
`cpu_time_system`	The amount of time that the CPU is in system mode. This metric is measured in hundredths of a second. Unit: None

Metric	Description
`cpu_time_user`	The amount of time that the CPU is in user mode. This metric is measured in hundredths of a second. Unit: None
`cpu_usage_active`	The percentage of time that the CPU is active in any capacity. Unit: Percent
`cpu_usage_guest`	The percentage of time that the CPU is running a virtual CPU for a guest operating system. Unit: Percent
`cpu_usage_guest_nice`	The percentage of time that the CPU is running a virtual CPU for a guest operating system, which is low-priority and can be interrupted by other processes. Unit: Percent
`cpu_usage_idle`	The percentage of time that the CPU is idle. Unit: Percent
`cpu_usage_iowait`	The percentage of time that the CPU is waiting for I/O operations to complete. Unit: Percent
`cpu_usage_irq`	The percentage of time that the CPU is servicing interrupts. Unit: Percent

Metric	Description
cpu_usage_nice	The percentage of time that the CPU is in user mode with low-priority processes, which higher-pr iority processes can easily interrupt. Unit: Percent
cpu_usage_softirq	The percentage of time that the CPU is servicing software interrupts. Unit: Percent
cpu_usage_steal	The percentage of time that the CPU is in *stolen time*, or time spent in other operating systems in a virtualized environment. Unit: Percent
cpu_usage_system	The percentage of time that the CPU is in system mode. Unit: Percent
cpu_usage_user	The percentage of time that the CPU is in user mode. Unit: Percent
disk_free	Free space on the disks. Unit: Bytes
disk_inodes_free	The number of available index nodes on the disk. Unit: Count
disk_inodes_total	The total number of index nodes reserved on the disk. Unit: Count

Metric	Description
`disk_inodes_used`	The number of used index nodes on the disk. Unit: Count
`disk_total`	Total space on the disks, including used and free. Unit: Bytes
`disk_used`	Used space on the disks. Unit: Bytes
`disk_used_percent`	The percentage of total disk space that is used. Unit: Percent
`diskio_iops_in_progress`	The number of I/O requests that have been issued to the device driver but have not yet completed. Unit: Count
`diskio_io_time`	The amount of time that the disk has had I/O requests queued. Unit: Milliseconds The only statistic that should be used for this metric is `Sum`. Do not use `Average`.
`diskio_reads`	The number of disk read operations. Unit: Count The only statistic that should be used for this metric is `Sum`. Do not use `Average`.

Metric	Description
`diskio_read_bytes`	The number of bytes read from the disks. Unit: Bytes The only statistic that should be used for this metric is `Sum`. Do not use `Average`.
`diskio_read_time`	The amount of time that read requests have waited on the disks. Multiple read requests waiting at the same time increase the number. For example, if 5 requests all wait for an average of 100 milliseconds, 500 is reported. Unit: Milliseconds The only statistic that should be used for this metric is `Sum`. Do not use `Average`.
`diskio_writes`	The number disk write operations. Unit: Count The only statistic that should be used for this metric is `Sum`. Do not use `Average`.
`diskio_write_bytes`	The number of bytes written to the disks. Unit: Bytes The only statistic that should be used for this metric is `Sum`. Do not use `Average`.

Metric	Description
diskio_write_time	The amount of time that write requests have waited on the disks. Multiple write requests waiting at the same time increase the number. For example, if 8 requests all wait for an average of 1000 milliseco nds, 8000 is reported. Unit: Milliseconds The only statistic that should be used for this metric is `Sum`. Do not use `Average`.
ethtool_bw_in_allowance_exc eeded	The number of packets queued and/or dropped because the inbound aggregate bandwidth exceeded the maximum for the instance. This metric is collected only if you have listed it in the `ethtool` subsection of the `metrics_c ollected` section of the CloudWatch agent configuration file. For more information, see [Collect network performance metrics] Unit: None
ethtool_bw_out_allowance_ex ceeded	The number of packets queued and/or dropped because the outbound aggregate bandwidth exceeded the maximum for the instance. This metric is collected only if you have listed it in the `ethtool` subsection of the `metrics_c ollected` section of the CloudWatch agent configuration file. For more information, see [Collect network performance metrics] Unit: None

Metric	Description
`ethtool_conntrack_allowance_exceeded`	The number of packets dropped because connection tracking exceeded the maximum for the instance and new connections could not be established. This can result in packet loss for traffic to or from the instance. This metric is collected only if you have listed it in the `ethtool` subsection of the `metrics_collected` section of the CloudWatch agent configuration file. For more information, see Collect network performance metrics Unit: None
`ethtool_linklocal_allowance_exceeded`	The number of packets dropped because the PPS of the traffic to local proxy services exceeded the maximum for the network interface. This impacts traffic to the DNS service, the Instance Metadata Service, and the Amazon Time Sync Service. This metric is collected only if you have listed it in the `ethtool` subsection of the `metrics_collected` section of the CloudWatch agent configuration file. For more information, see Collect network performance metrics Unit: None

Metric	Description
`ethtool_pps_allowance_exceeded`	The number of packets queued and/or dropped because the bidirectional PPS exceeded the maximum for the instance. This metric is collected only if you have listed it in the `ethtool` subsection of the `metrics_collected` section of the CloudWatch agent configuration file. For more information, see [Collect network performance metrics](). Unit: None
`mem_active`	The amount of memory that has been used in some way during the last sample period. Unit: Bytes
`mem_available`	The amount of memory that is available and can be given instantly to processes. Unit: Bytes
`mem_available_percent`	The percentage of memory that is available and can be given instantly to processes. Unit: Percent
`mem_buffered`	The amount of memory that is being used for buffers. Unit: Bytes
`mem_cached`	The amount of memory that is being used for file caches. Unit: Bytes

Metric	Description
mem_free	The amount of memory that isn't being used. Unit: Bytes
mem_inactive	The amount of memory that hasn't been used in some way during the last sample period Unit: Bytes
mem_total	The total amount of memory. Unit: Bytes
mem_used	The amount of memory currently in use. Unit: Bytes
mem_used_percent	The percentage of memory currently in use. Unit: Percent
net_bytes_recv	The number of bytes received by the network interface. Unit: Bytes The only statistic that should be used for this metric is Sum. Do not use Average.
net_bytes_sent	The number of bytes sent by the network interface. Unit: Bytes The only statistic that should be used for this metric is Sum. Do not use Average.

Metric	Description
`net_drop_in`	The number of packets received by this network interface that were dropped. Unit: Count The only statistic that should be used for this metric is `Sum`. Do not use `Average`.
`net_drop_out`	The number of packets transmitted by this network interface that were dropped. Unit: Count The only statistic that should be used for this metric is `Sum`. Do not use `Average`.
`net_err_in`	The number of receive errors detected by this network interface. Unit: Count The only statistic that should be used for this metric is `Sum`. Do not use `Average`.
`net_err_out`	The number of transmit errors detected by this network interface. Unit: Count The only statistic that should be used for this metric is `Sum`. Do not use `Average`.
`net_packets_sent`	The number of packets sent by this network interface. Unit: Count The only statistic that should be used for this metric is `Sum`. Do not use `Average`.

Metric	Description
`net_packets_recv`	The number of packets received by this network interface. Unit: Count The only statistic that should be used for this metric is `Sum`. Do not use `Average`.
`netstat_tcp_close`	The number of TCP connections with no state. Unit: Count
`netstat_tcp_close_wait`	The number of TCP connections waiting for a termination request from the client. Unit: Count
`netstat_tcp_closing`	The number of TCP connections that are waiting for a termination request with acknowledgment from the client. Unit: Count
`netstat_tcp_established`	The number of TCP connections established. Unit: Count
`netstat_tcp_fin_wait1`	The number of TCP connections in the `FIN_WAIT1` state during the process of closing a connection. Unit: Count
`netstat_tcp_fin_wait2`	The number of TCP connections in the `FIN_WAIT2` state during the process of closing a connection. Unit: Count

Metric	Description
`netstat_tcp_last_ack`	The number of TCP connections waiting for the client to send acknowledgment of the connectio n termination message. This is the last state right before the connection is closed down. Unit: Count
`netstat_tcp_listen`	The number of TCP ports currently listening for a connection request. Unit: Count
`netstat_tcp_none`	The number of TCP connections with inactive clients. Unit: Count
`netstat_tcp_syn_sent`	The number of TCP connections waiting for a matching connection request after having sent a connection request. Unit: Count
`netstat_tcp_syn_recv`	The number of TCP connections waiting for connection request acknowledgment after having sent and received a connection request. Unit: Count
`netstat_tcp_time_wait`	The number of TCP connections currently waiting to ensure that the client received the acknowledgment of its connection termination request. Unit: Count
`netstat_udp_socket`	The number of current UDP connections. Unit: Count

Metric	Description
processes_blocked	The number of processes that are blocked. Unit: Count
processes_dead	The number of processes that are dead, indicated by the X state code on Linux. This metric is not collected on macOS computers. Unit: Count
processes_idle	The number of processes that are idle (sleeping for more than 20 seconds). Available only on FreeBSD instances. Unit: Count
processes_paging	The number of processes that are paging, indicated by the W state code on Linux. This metric is not collected on macOS computers. Unit: Count
processes_running	The number of processes that are running, indicated by the R state code. Unit: Count
processes_sleeping	The number of processes that are sleeping, indicated by the S state code. Unit: Count
processes_stopped	The number of processes that are stopped, indicated by the T state code. Unit: Count

Metric	Description
`processes_total`	The total number of processes on the instance. Unit: Count
`processes_total_threads`	The total number of threads making up the processes. This metric is available only on Linux instances. This metric is not collected on macOS computers. Unit: Count
`processes_wait`	The number of processes that are paging, indicated by the `W` state code on FreeBSD instances. This metric is available only on FreeBSD instances, and is not available on Linux, Windows Server, or macOS instances. Unit: Count
`processes_zombies`	The number of zombie processes, indicated by the `Z` state code. Unit: Count
`swap_free`	The amount of swap space that isn't being used. Unit: Bytes
`swap_used`	The amount of swap space currently in use. Unit: Bytes
`swap_used_percent`	The percentage of swap space currently in use. Unit: Percent

Definitions of memory metrics collected by the CloudWatch agent

When the CloudWatch agent collects memory metrics, the source is the host's memory management subsystem. For example, the Linux kernel exposes OS-maintained data in `/proc`. For memory, the data is in `/proc/meminfo`.

Each different operating system and architecture has different calculations of the resources that are used by processes. For more information, see the following sections.

During each collection interval, the CloudWatch agent on each instance collects the instance resources and calculates the resources being used by all processes which are running in that instance. This information is reported back to CloudWatch metrics. You can configure the length of the collection interval in the CloudWatch agent configuration file. For more information, see CloudWatch agent configuration file: Agent section.

The following list explains how the memory metrics that the CloudWatch agent collects are defined.

- **Active Memory**– Memory that is being used by a process. In other words, the memory used by current running apps.
- **Available Memory**– The memory that can be instantly given to the processes without the system going into swap (also known as virtual memory).
- **Buffer Memory**– The data area shared by hardware devices or program processes that operate at different speeds and priorities.
- **Cached Memory**– Stores program instructions and data that are used repeatedly in the operation of programs that the CPU is likely to need next.
- **Free Memory**– Memory that is not being used at all and is readily available. It is completely free for the system to be used when needed.
- **Inactive Memory**– Pages that have not been accessed "recently".
- **Total Memory**– The size of the actual physical memory RAM.
- **Used Memory**– Memory that is currently in use by programs and processes.

Topics

- [Windows: Metrics collected]
- [Example: Calculating memory metrics on Linux]

Linux: Metrics collected and calculations used

Metrics collected and units:

- Active (Bytes)
- Available (Bytes)
- Available Percent (Percent)
- Buffered (Bytes)
- Cached (Bytes)
- Free (Bytes)
- Inactive (Bytes)
- Total (Bytes)
- Used (Bytes)
- Used Percent (Percent)

Used memory = Total Memory - Free Memory - Cached memory - Buffer memory

Total memory = Used Memory + Free Memory + Cached memory + Buffer memory

macOS: Metrics collected and calculations used

Metrics collected and units:

- Active (Bytes)
- Available (Bytes)
- Available Percent (Percent)
- Free (Bytes)
- Inactive (Bytes)
- Total (Bytes)
- Used (Bytes)
- Used Percent (Percent)

Available memory = Free Memory + Inactive memory

Used memory = Total Memory - Available memory

Total memory = Available Memory + Used Memory

Windows: Metrics collected

The metrics collected on Windows hosts are listed below. All of these metrics have None for `Unit`.

- Available bytes
- Cache Faults/sec
- Page Faults/sec
- Pages/sec

There are no calculations used for Windows metrics because the CloudWatch agent parses events from performance counters.

Example: Calculating memory metrics on Linux

As an example, suppose that entering the **`cat /proc/meminfo`** command on a Linux host shows the following results:

```
MemTotal:       3824388 kB
MemFree:         462704 kB
MemAvailable:   2157328 kB
Buffers:         126268 kB
Cached:         1560520 kB
SReclaimable:    289080 kB>
```

In this example, the CloudWatch agent will collect the following values. All the values that the CloudWatch agent collects and reports are in bytes.

- `mem_total`: 3916173312 bytes
- `mem_available`: 2209103872 bytes (MemFree + Cached)
- `mem_free`: 473808896 bytes
- `mem_cached`: 1893990400 bytes (`cached` + `SReclaimable`
- `mem_used`: 1419075584 bytes (`MemTotal` – (`MemFree` + `Buffers` + (`Cached` + `SReclaimable`)))

- `mem_buffered`: 129667072 bytes
- `mem_available_percent`: 56.41%
- `mem_used_percent`: 36.24% (`mem_used` / `mem_total`) * 100

Using the CloudWatch agent with related telemetry

Metrics and logs that are sent to CloudWatch can include an optional entity to correlate telemetry. Entities are used in the Explore related pane. The CloudWatch agent sends entities with a service name and environment name included.

The agent chooses the service name and environment name from the following data.

Service name

The agent chooses the service name from the following options, in priority order:

- **Application Signals instrumentation** – The agent sends the service name used by Application Signals. This can be overwritten by changing the `OTEL_SERVICE_NAME` environment variable used by supported OpenTelemetry instrumentation libraries.
- **CloudWatch agent configuration** – You can configure the agent to use a specific service name. This can be configured at the agent, plugin, metrics, logs, or log file level.
- **Kubernetes workload name** – For Kubernetes workloads, the agent sends the name of the workload for the corresponding pod, in the following priority order.
  - Deployment name
  - ReplicaSet name
  - StatefulSet name
  - DaemonSet name
  - CronJob name
  - Job name
  - Pod name
  - Container name
- **Resource tags from instance metadata** – For Amazon EC2 workloads, the agent sends the a name from tags, in the following order.
  - service

  - application
  - app

  You must setup instance metadata for the agent to be able to access tags.

- **Default** – If no other service name is found, the agent will send the name `Unknown`.

Environment name

The agent chooses the environment name from the following options, in priority order:

- **Application Signals instrumentation** – The agent sends the environment name used by Application Signals. This can be overwritten by setting a `deployment.environment` environment variable used by supported OpenTelemetry instrumentation libraries. For example, applications may set the environment variable `OTEL_RESOURCE_ATTRIBUTES=deployment.environment=MyEnvironment`.
- **CloudWatch agent configuration** – You can configure the agent to use a specific environment name. This can be configured at the agent, plugin, metrics, logs, or log file level.
- **Cluster name and workspace** – For Amazon EKS, `eks:`*`cluster-name`*`/`*`Namespace`*. For native Kubernetes running on Amazon EC2, `k8s:`*`cluster-name`*`/`*`Namespace`*.
- **Resource tags from instance metadata** – For Amazon EC2 workloads, the agent can will use the `AutoScalingGroup` tag.

  You must setup instance metadata for the agent to be able to access tags.

- By default, Amazon EC2 instances that aren't running Kubernetes will get the environment name `ec2:default`.

Common scenarios with the CloudWatch agent

This section provides you with different scenarios that outline how to complete common configuration and customization tasks for the CloudWatch agent.

Topics

- Multiple CloudWatch agent configuration files
- Aggregating or rolling up metrics collected by the CloudWatch agent
- Collecting high-resolution metrics with the CloudWatch agent
- Sending metrics, logs, and traces to a different account
- Timestamp differences between the unified CloudWatch agent and the earlier CloudWatch Logs agent
- Appending OpenTelemetry collector configuration files

Running the CloudWatch agent as a different user

On Linux servers, the CloudWatch runs as the root user by default. To have the agent run as a different user, use the `run_as_user` parameter in the `agent` section in the CloudWatch agent configuration file. This option is available only on Linux servers.

If you're already running the agent with the root user and want to change to using a different user, use one of the following procedures.

To run the CloudWatch agent as a different user on an EC2 instance running Linux

1. Download and install a new CloudWatch agent package. For more information, see Download the CloudWatch agent package.
2. Create a new Linux user or use the default user named `cwagent` that the RPM or DEB file created.
3. Provide credentials for this user in one of these ways:
   - If the file `.aws/credentials` exists in the home directory of the root user, you must create a credentials file for the user you are going to use to run the CloudWatch agent. This credentials file will be `/home/`*`username`*`/.aws/credentials`. Then set the value of the `shared_credential_file` parameter in `common-config.toml` to the pathname of the credential file. For more information, see (Optional) Modify the common configuration for proxy or Region information.
   - If the file `.aws/credentials` does not exist in the home directory of the root user, you can do one of the following:
     - Create a credentials file for the user you are going to use to run the CloudWatch agent. This credentials file will be `/home/`*`username`*`/.aws/credentials`. Then set the value of the `shared_credential_file` parameter in `common-config.toml`

to the pathname of the credential file. For more information, see (Optional) Modify the common configuration for proxy or Region information.

- Instead of creating a credentials file, attach an IAM role to the instance. The agent uses this role as the credential provider.

4. In the CloudWatch agent configuration file, add the following line in the `agent` section:

```
"run_as_user": "username"
```

Make other modifications to the configuration file as needed. For more information, see Create the CloudWatch agent configuration file

5. Give the user the required permissions. The user must have Read (r) permissions for the log files to be collected, and must have Execute (x) permission on every directory in the log files' path.

6. Start the agent with the configuration file that you just modified.

```
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-config -m ec2 -s -c file:configuration-file-path
```

To run the CloudWatch agent as a different user on an on-premises server running Linux

1. Download and install a new CloudWatch agent package. For more information, see Download the CloudWatch agent package.

2. Create a new Linux user or use the default user named `cwagent` that the RPM or DEB file created.

3. Store the credentials of this user to a path that the user can access, such as `/home/username/.aws/credentials`.

4. Set the value of the `shared_credential_file` parameter in `common-config.toml` to the pathname of the credential file. For more information, see (Optional) Modify the common configuration for proxy or Region information.

5. In the CloudWatch agent configuration file, add the following line in the `agent` section:

```
"run_as_user": "username"
```

Make other modifications to the configuration file as needed. For more information, see Create the CloudWatch agent configuration file

6. Give the user required permissions. The user must have Read (r) permissions for the log files to be collected, and must have Execute (x) permission on every directory in the log files' path.
7. Start the agent with the configuration file that you just modified.

```
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-config -m ec2 -s -c file:configuration-file-path
```

How the CloudWatch agent handles sparse log files

Sparse files are files with both empty blocks and real contents. A sparse file uses disk space more efficiently by writing brief information representing the empty blocks to disk instead of the actual null bytes which makes up the block. This makes the actual size of a sparse file usually much smaller than its apparent size.

However, the CloudWatch agent doesn’t treat sparse files differently than it treats normal files. When the agent reads a sparse file, the empty blocks are treated as "real" blocks filled with null bytes. Because of this, the CloudWatch agent publishes as many bytes as the apparent size of a sparse file to CloudWatch.

Configuring the CloudWatch agent to publish a sparse file can cause higher than expected CloudWatch costs, so we recommend not to do so. For example, `/var/logs/lastlog` in Linux is usually a very sparse file, and we recommend that you don't publish it to CloudWatch.

Adding custom dimensions to metrics collected by the CloudWatch agent

To add custom dimensions such as tags to metrics collected by the agent, add the `append_dimensions` field to the section of the agent configuration file that lists those metrics.

For example, the following example section of the configuration file adds a custom dimension named `stackName` with a value of `Prod` to the `cpu` and `disk` metrics collected by the agent.

```
"cpu":{
  "resources":[
    "*"
  ],
  "measurement":[
    "cpu_usage_guest",
```

```
      "cpu_usage_nice",
      "cpu_usage_idle"
    ],
    "totalcpu":false,
    "append_dimensions":{
      "stackName":"Prod"
    }
  },
  "disk":{
    "resources":[
      "/",
      "/tmp"
    ],
    "measurement":[
      "total",
      "used"
    ],
    "append_dimensions":{
      "stackName":"Prod"
    }
  }
```

Remember that any time you change the agent configuration file, you must restart the agent to have the changes take effect.

Multiple CloudWatch agent configuration files

On both Linux servers and Windows servers, you can set up the CloudWatch agent to use multiple configuration files. For example, you can use a common configuration file that collects a set of metrics, logs, and traces that you always want to collect from all servers in your infrastructure. You can then use additional configuration files that collect metrics from certain applications or in certain situations.

To set this up, first create the configuration files that you want to use. Any configuration files that will be used together on the same server must have different file names. You can store the configuration files on servers or in Parameter Store.

Start the CloudWatch agent using the `fetch-config` option and specify the first configuration file. To append the second configuration file to the running agent, use the same command but with the `append-config` option. All metrics, logs, and traces listed in either configuration file are collected. The following example commands illustrate this scenario using configurations stores as files. The first line starts the agent using the `infrastructure.json` configuration file, and the second line appends the `app.json` configuration file.

The following example commands are for Linux.

```
/opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-config -m ec2
 -s -c file:/tmp/infrastructure.json
```

```
/opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a append-config -m
 ec2 -s -c file:/tmp/app.json
```

The following example commands are for Windows Server.

```
& "C:\Program Files\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1"
 -a fetch-config -m ec2 -s -c file:"C:\Program Files\Amazon\AmazonCloudWatchAgent
\infrastructure.json"
```

```
& "C:\Program Files\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1"
 -a append-config -m ec2 -s -c file:"C:\Program Files\Amazon\AmazonCloudWatchAgent
\app.json"
```

The following example configuration files illustrate a use for this feature. The first configuration file is used for all servers in the infrastructure, and the second collects only logs from a certain application and is appended to servers running that application.

infrastructure.json

```
{
  "metrics": {
    "metrics_collected": {
      "cpu": {
        "resources": [
          "*"
        ],
        "measurement": [
          "usage_active"
        ],
        "totalcpu": true
      },
      "mem": {
         "measurement": [
```

```
            "used_percent"
          ]
        }
      }
    },
    "logs": {
      "logs_collected": {
        "files": {
          "collect_list": [
            {
              "file_path": "/opt/aws/amazon-cloudwatch-agent/logs/amazon-cloudwatch-
agent.log",
              "log_group_name": "amazon-cloudwatch-agent.log"
            },
            {
              "file_path": "/var/log/messages",
              "log_group_name": "/var/log/messages"
            }
          ]
        }
      }
    }
}
```

app.json

```
{
    "logs": {
        "logs_collected": {
            "files": {
                "collect_list": [
                    {
                        "file_path": "/app/app.log*",
                        "log_group_name": "/app/app.log"
                    }
                ]
            }
        }
    }
}
```

Any configuration files appended to the configuration must have different file names from each other and from the initial configuration file. If you use `append-config` with a configuration file with the same file name as a configuration file that the agent is already using, the append command overwrites the information from the first configuration file instead of appending to it. This is true even if the two configuration files with the same file name are on different file paths.

The preceding example shows the use of two configuration files, but there is no limit to the number of configuration files that you can append to the agent configuration. You can also mix the use of configuration files located on servers and configurations located in Parameter Store.

Aggregating or rolling up metrics collected by the CloudWatch agent

To aggregate or roll up metrics collected by the agent, add an `aggregation_dimensions` field to the section for that metric in the agent configuration file.

For example, the following configuration file snippet rolls up metrics on the `AutoScalingGroupName` dimension. The metrics from all instances in each Auto Scaling group are aggregated and can be viewed as a whole.

```
"metrics": {
  "cpu":{...}
  "disk":{...}
  "aggregation_dimensions" : [["AutoScalingGroupName"]]
}
```

To roll up along the combination of each `InstanceId` and `InstanceType` dimensions in addition to rolling up on the Auto Scaling group name, add the following.

```
"metrics": {
  "cpu":{...}
  "disk":{...}
  "aggregation_dimensions" : [["AutoScalingGroupName"], ["InstanceId", "InstanceType"]]
}
```

To roll up metrics into one collection instead, use `[]`.

```
"metrics": {
  "cpu":{...}
  "disk":{...}
  "aggregation_dimensions" : [[]]
}
```

Remember that any time you change the agent configuration file, you must restart the agent to have the changes take effect.

Collecting high-resolution metrics with the CloudWatch agent

The `metrics_collection_interval` field specifies the time interval for the metrics collected, in seconds. By specifying a value of less than 60 for this field, the metrics are collected as high-resolution metrics.

For example, if your metrics should all be high-resolution and collected every 10 seconds, specify 10 as the value for `metrics_collection_interval` under the `agent` section as a global metrics collection interval.

```
"agent": {
  "metrics_collection_interval": 10
}
```

Alternatively, the following example sets the `cpu` metrics to be collected every second, and all other metrics are collected every minute.

```
"agent":{
  "metrics_collection_interval": 60
},
"metrics":{
  "metrics_collected":{
    "cpu":{
      "resources":[
        "*"
      ],
      "measurement":[
        "cpu_usage_guest"
      ],
      "totalcpu":false,
      "metrics_collection_interval": 1
    },
    "disk":{
      "resources":[
        "/",
        "/tmp"
      ],
      "measurement":[
        "total",
```

```
        "used"
      ]
    }
  }
}
```

Remember that any time you change the agent configuration file, you must restart the agent to have the changes take effect.

Sending metrics, logs, and traces to a different account

To have the CloudWatch agent send the metrics, logs, or traces to a different account, specify a `role_arn` parameter in the agent configuration file on the sending server. The `role_arn` value specifies an IAM role in the target account that the agent uses when sending data to the target account. This role enables the sending account to assume a corresponding role in the target account when delivering the metrics or logs to the target account.

You can also specify separate `role_arn` strings in the agent configuration file: one to use when sending metrics, another for sending logs, and another for sending traces.

The following example of part of the `agent` section of the configuration file sets the agent to use `CrossAccountAgentRole` when sending data to a different account.

```
{
  "agent": {
    "credentials": {
      "role_arn": "arn:aws:iam::123456789012:role/CrossAccountAgentRole"
    }
  },
  .....
}
```

Alternatively, the following example sets different roles for the sending account to use for sending metrics, logs, and traces:

```

"metrics": {
    "credentials": {
     "role_arn": "RoleToSendMetrics"
    },
    "metrics_collected": {....
```

```
"logs": {
    "credentials": {
    "role_arn": "RoleToSendLogs"
    },
    ....
```

Policies needed

When you specify a `role_arn` in the agent configuration file, you must also make sure the IAM roles of the sending and target accounts have certain policies. The roles in both the sending and target accounts should have `CloudWatchAgentServerPolicy`. For more information about assigning this policy to a role, see [Create IAM roles to use with the CloudWatch agent on Amazon EC2 instances](#).

The role in the sending account also must include the following policy. You add this policy on the **Permissions** tab in the IAM console when you edit the role.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sts:AssumeRole"
            ],
            "Resource": [
                "arn:aws:iam::target-account-ID:role/agent-role-in-target-account"
            ]
        }
    ]
}
```

The role in the target account must include the following policy so that it recognizes the IAM role used by the sending account. You add this policy on the **Trust relationships** tab in the IAM console when you edit the role. The role in the target account where you add this policy is the role you created in [Create IAM roles and users for use with CloudWatch agent](#). This role is the role specified in *`agent-role-in-target-account`* in the policy used by the sending account.

```
{
  "Version": "2012-10-17",
```

```
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": [
          "arn:aws:iam::sending-account-ID:role/role-in-sender-account"
        ]
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Timestamp differences between the unified CloudWatch agent and the earlier CloudWatch Logs agent

The CloudWatch agent supports a different set of symbols for timestamp formats, compared to the earlier CloudWatch Logs agent. These differences are shown in the following table.

Symbols supported by both agents	Symbols supported only by unified CloudWatch agent	Symbols supported only by earlier CloudWatch Logs agent
%A, %a, %b, %B, %d, %f, %H, %l, %m, %M, %p, %S, %y, %Y, %Z, %z	%-d, %-l, %-m, %-M, %-S	%c,%j, %U, %W, %w

For more information about the meanings of the symbols supported by the new CloudWatch agent, see CloudWatch Agent Configuration File: Logs Section in the *Amazon CloudWatch User Guide*. For information about symbols supported by the CloudWatch Logs agent, see Agent Configuration File in the *Amazon CloudWatch Logs User Guide*.

Appending OpenTelemetry collector configuration files

The CloudWatch agent supports supplemental OpenTelemetry collector configuration files alongside its own configuration files. This feature allows you to use CloudWatch agent features such as CloudWatch Application Signals or Container Insights through the CloudWatch agent configuration and bring in your existing OpenTelemetry collector configuration with a single agent.

To prevent merge conflicts with pipelines automatically created by CloudWatch agent, we recommend that you add a custom suffix to each of the components and pipelines in your OpenTelemetry collector configuration.

```
receivers:
  otlp/custom-suffix:
    protocols:
      http:

exporters:
  awscloudwatchlogs/custom-suffix:
    log_group_name: "test-group"
    log_stream_name: "test-stream"

service:
  pipelines:
    logs/custom-suffix:
      receivers: [otlp/custom-suffix]
      exporters: [awscloudwatchlogs/custom-suffix]
```

To configure the CloudWatch agent, start the CloudWatch agent using the `fetch-config` option and specify the CloudWatch agent's configuration file. CloudWatch agent requires at least one CloudWatch agent configuration file.

```
/opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-config -c
 file:/tmp/agent.json -s
```

Next, use the `append-config` option while specifying the OpenTelemetry collector configuration file.

```
/opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a append-config -c
 file:/tmp/otel.yaml -s
```

The agent merges the two configuration files on start up and logs the resolved configuration.

Troubleshooting the CloudWatch agent

You can use the information in this section to troubleshoot issues you might encounter with the CloudWatch agent.

Topics

CloudWatch agent command line parameters

To see the full list of parameters supported by the CloudWatch agent, enter the following at the command line at a computer where you have it installed:

```
amazon-cloudwatch-agent-ctl -help
```

Install the CloudWatch agent using Run Command fails

To install the CloudWatch agent using Systems Manager Run Command, the SSM Agent on the target server must be version 2.2.93.0 or later of the SSM Agent agent. If your SSM Agent isn't the correct version, you might see errors that include the following messages:

```
no latest version found for package AmazonCloudWatchAgent on platform linux
```

```
failed to download installation package reliably
```

For information about updating your SSM Agent version, see Installing and Configuring SSM Agent in the *Amazon Systems Manager User Guide*.

The CloudWatch agent won't start

If the CloudWatch agent fails to start, there might be an issue in your configuration. Configuration information is logged in the `configuration-validation.log` file. This file is located in `/opt/aws/amazon-cloudwatch-agent/logs/configuration-validation.log` on Linux servers and in `$Env:ProgramData\Amazon\AmazonCloudWatchAgent\Logs\configuration-validation.log` on servers running Windows Server.

Verify that the CloudWatch agent is running

You can query the CloudWatch agent to find whether it's running or stopped. You can use Amazon Systems Manager to do this remotely. You can also use the command line, but only to check the local server.

To query the status of the CloudWatch agent using Run Command

1. Open the Systems Manager console at https://console.amazonaws.cn/systems-manager/.
2. In the navigation pane, choose **Run Command**.

   -or-

   If the Amazon Systems Manager home page opens, scroll down and choose **Explore Run Command**.
3. Choose **Run command**.
4. In the **Command document** list, choose the button next to **AmazonCloudWatch-ManageAgent**.
5. In the **Action** list, choose **status**.
6. For **Optional Configuration Source** choose **default** and keep **Optional Configuration Location** blank.
7. In the **Target** area, choose the instance to check.
8. Choose **Run**.

If the agent is running, the output resembles the following.

```
{
       "status": "running",
       "starttime": "2017-12-12T18:41:18",
       "version": "1.73.4"
```

```
}
```

If the agent is stopped, the "status" field displays "stopped".

To query the status of the CloudWatch agent locally using the command line

- On a Linux server, enter the following:

  ```
  sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -m ec2 -a
   status
  ```

  On a server running Windows Server, enter the following in PowerShell as an administrator:

  ```
  & $Env:ProgramFiles\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1 -m
   ec2 -a status
  ```

The CloudWatch agent won't start, and the error mentions an Amazon EC2 Region

If the agent doesn't start and the error message mentions an Amazon EC2 Region endpoint, you might have configured the agent to need access to the Amazon EC2 endpoint without granting that access.

For example, if you specify a value for the append_dimensions parameter in the agent configuration file that depends on Amazon EC2 metadata and you use proxies, you must make sure that the server can access the endpoint for Amazon EC2. For more information about these endpoints, see Amazon Elastic Compute Cloud (Amazon EC2) in the *Amazon Web Services General Reference*.

The CloudWatch agent won't start on Windows Server

On Windows Server, you might see the following error:

```
Start-Service : Service 'Amazon CloudWatch Agent (AmazonCloudWatchAgent)' cannot be
 started due to the following
error: Cannot start service AmazonCloudWatchAgent on computer '.'.
At C:\Program Files\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1:113
 char:12
+     $svc | Start-Service
```

```
+             ~~~~~~~~~~~~~
    + CategoryInfo          : OpenError:
 (System.ServiceProcess.ServiceController:ServiceController) [Start-Service],
   ServiceCommandException
    + FullyQualifiedErrorId :
 CouldNotStartService,Microsoft.PowerShell.Commands.StartServiceCommand
```

To fix this, first make sure that the server service is running. This error can be seen if the agent tries to start when the server service isn't running.

If the server service is already running, the following may be the issue. On some Windows Server installations, the CloudWatch agent takes more than 30 seconds to start. Because Windows Server, by default, allows only 30 seconds for services to start, this causes the agent to fail with an error similar to the following:

To fix this issue, increase the service timeout value. For more information, see [A service does not start, and events 7000 and 7011 are logged in the Windows event log].

Where are the metrics?

If the CloudWatch agent has been running but you can't find metrics collected by it in the Amazon Web Services Management Console or the Amazon CLI, confirm that you're using the correct namespace. By default, the namespace for metrics collected by the agent is `CWAgent`. You can customize this namespace using the `namespace` field in the `metrics` section of the agent configuration file. If you don't see the metrics that you expect, check the configuration file to confirm the namespace being used.

When you first download the CloudWatch agent package, the agent configuration file is `amazon-cloudwatch-agent.json`. This file is in the directory where you ran the configuration wizard, or you might have moved it to a different directory. If you use the configuration wizard, the agent configuration file output from the wizard is named `config.json`. For more information about the configuration file, including the `namespace` field, see [CloudWatch agent configuration file: Metrics section].

The CloudWatch agent takes a long time to run in a container or logs a hop limit error

When you run the CloudWatch agent as a container service and want to add Amazon EC2 metric dimensions to all metrics collected by the agent, you might see the following errors in version v1.247354.0 of the agent:

```
2022-06-07T03:36:11Z E! [processors.ec2tagger] ec2tagger: Unable to retrieve Instance
 Metadata Tags. This plugin must only be used on an EC2 instance.
2022-06-07T03:36:11Z E! [processors.ec2tagger] ec2tagger: Please increase hop limit
 to 2 by following this document https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/
configuring-instance-metadata-options.html#configuring-IMDS-existing-instances.
2022-06-07T03:36:11Z E! [telegraf] Error running agent: could not initialize processor
 ec2tagger: EC2MetadataRequestError: failed to get EC2 instance identity document
caused by: EC2MetadataError: failed to make EC2Metadata request
        status code: 401, request id:
caused by:
```

You might see this error if the agent tries to get metadata from IMDSv2 inside a container without an appropriate hop limit. In versions of the agent earlier than v1.247354.0, you can experience this issue without seeing the log message.

To solve this, increase the hop limit to 2 by following the instructions in [Configure the instance metadata options](#).

I updated my agent configuration but don’t see the new metrics or logs in the CloudWatch console

If you update your CloudWatch agent configuration file, the next time that you start the agent, you need to use the **`fetch-config`** option. For example, if you stored the updated file on the local computer, enter the following command:

```
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-config -
s -m ec2 -c file:configuration-file-path
```

CloudWatch agent files and locations

The following table lists the files installed by and used with the CloudWatch agent, along with their locations on servers running Linux or Windows Server.

File	Linux location	Windows Server location
The control script that controls starting, stopping, and restarting the agent.	`/opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl` or `/usr/bin/`	`$Env:ProgramFiles\Amazon\AmazonCloudWatchAgent\amazon-`

File	Linux location	Windows Server location
	`amazon-cloudwatch-agent-ctl`	`cloudwatch-agent-ctl.ps1`
The log file the agent writes to. You might need to attach this when contacting Amazon Web Services Support.	`/opt/aws/amazon-cloudwatch-agent/logs/amazon-cloudwatch-agent.log` or `/var/log/amazon/amazon-cloudwatch-agent/amazon-cloudwatch-agent.log`	`$Env:ProgramData\Amazon\AmazonCloudWatchAgent\Logs\amazon-cloudwatch-agent.log`
Agent configuration validation file.	`/opt/aws/amazon-cloudwatch-agent/logs/configuration-validation.log` or `/var/log/amazon/amazon-cloudwatch-agent/configuration-validation.log`	`$Env:ProgramData\Amazon\AmazonCloudWatchAgent\Logs\configuration-validation.log`
The JSON file used to configure the agent immediately after the wizard creates it. For more information, see [Create the CloudWatch agent configuration file]().	`/opt/aws/amazon-cloudwatch-agent/bin/config.json`	`$Env:ProgramFiles\Amazon\AmazonCloudWatchAgent\config.json`
The JSON file used to configure the agent if this configuration file has been downloaded from Parameter Store.	`/opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.json` or `/etc/amazon/amazon-cloudwatch-agent/amazon-cloudwatch-agent.json`	`$Env:ProgramData\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent.json`

File	Linux location	Windows Server location
The TOML file used to specify Region and credential information to be used by the agent, overriding system defaults.	`/opt/aws/amazon-cloudwatch-agent/etc/common-config.toml` or `/etc/amazon/amazon-cloudwatch-agent/common-config.toml`	`$Env:ProgramData\Amazon\AmazonCloudWatchAgent\common-config.toml`
The TOML file that contains the converted contents of the JSON configuration file. The `amazon-cloudwatch-agent-ctl` script generates this file. Users should not directly modify this file. It can be useful for verifying that JSON to TOML translation was successful.	`/opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.toml` or `/etc/amazon/amazon-cloudwatch-agent/amazon-cloudwatch-agent.toml`	`$Env:ProgramData\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent.toml`
The YAML file that contains the converted contents of the JSON configuration file. The `amazon-cloudwatch-agent-ctl` script generates this file. You should not directly modify this file. This file can be useful for verifying that the JSON to YAML translation was successful.	`/opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.yaml or /etc/amazon/amazon-cloudwatch-agent/amazon-cloudwatch-agent.yaml`	`$Env:ProgramData\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent.yaml`

Logs generated by the CloudWatch agent

The agent generates a log while it runs. This log includes troubleshooting information. This log is the `amazon-cloudwatch-agent.log` file. This file is located in `/opt/aws/amazon-cloudwatch-agent/logs/amazon-cloudwatch-agent.log` on Linux servers and in `$Env:ProgramData\Amazon\AmazonCloudWatchAgent\Logs\amazon-cloudwatch-agent.log` on servers running Windows Server.

You can configure the agent to log additional details in the `amazon-cloudwatch-agent.log` file. In the agent configuration file, in the `agent` section, set the `debug` field to `true`, then reconfigure and restart the CloudWatch agent. To disable the logging of this extra information, set the `debug` field to `false`. Then, reconfigure and restart the agent. For more information, see [Manually create or edit the CloudWatch agent configuration file](#).

In versions 1.247350.0 and later of the CloudWatch agent, you can optionally set the `aws_sdk_log_level` field in the `agent` section of the agent configuration file to one or more of the following options. Separate multiple options with the `|` character.

- `LogDebug`
- `LogDebugWithSigning`
- `LogDebugWithHTTPBody`
- `LogDebugRequestRetries`
- `LogDebugWithEventStreamBody`

For more information about these options, see [LogLevelType](#).

Stopping and restarting the CloudWatch agent

You can manually stop the CloudWatch agent using either Amazon Systems Manager or the command line.

To stop the CloudWatch agent using Run Command

1. Open the Systems Manager console at https://console.amazonaws.cn/systems-manager/.
2. In the navigation pane, choose **Run Command**.

   -or-

If the Amazon Systems Manager home page opens, scroll down and choose **Explore Run Command**.

3. Choose **Run command**.
4. In the **Command document** list, choose **AmazonCloudWatch-ManageAgent**.
5. In the **Targets** area, choose the instance where you installed the CloudWatch agent.
6. In the **Action** list, choose **stop**.
7. Keep **Optional Configuration Source** and **Optional Configuration Location** blank.
8. Choose **Run**.

To stop the CloudWatch agent locally using the command line

- On a Linux server, enter the following:

```
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -m ec2 -a
 stop
```

On a server running Windows Server, enter the following in PowerShell as an administrator:

```
& $Env:ProgramFiles\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1 -m
 ec2 -a stop
```

To restart the agent, follow the instructions in (Optional) Modify the common configuration and named profile for CloudWatch agent.

Embedding metrics within logs

The CloudWatch embedded metric format allows you to generate custom metrics asynchronously in the form of logs written to CloudWatch Logs. You can embed custom metrics alongside detailed log event data, and CloudWatch automatically extracts the custom metrics so that you can visualize and alarm on them, for real-time incident detection. Additionally, the detailed log events associated with the extracted metrics can be queried using CloudWatch Logs Insights to provide deep insights into the root causes of operational events.

Embedded metric format helps you generate actionable custom metrics from ephemeral resources such as Lambda functions and containers. By using the embedded metric format to send logs from these ephemeral resources, you can now easily create custom metrics without having to instrument or maintain separate code, while gaining powerful analytical capabilities on your log data.

No setup is required to use the embedded metric format. Either structure your logs by following the Embedded metric format specification, or generate them using our client libraries and send them to CloudWatch Logs using the PutLogEvents API or the CloudWatch agent.

To generate metrics from logs with embedded metric format, you need the `logs:PutLogEvents` permission but you don't need to also have the `cloudwatch:PutMetricData` permission.

Charges are incurred for logs ingestion and archival, and custom metrics that are generated. For more information, see Amazon CloudWatch Pricing.

> **Note**
>
> Be careful when configuring your metric extraction as it impacts your custom metric usage and corresponding bill. If you unintentionally create metrics based on high-cardinality dimensions (such as `requestId`), the embedded metric format will by design create a custom metric corresponding to each unique dimension combination. For more information, see Dimensions.

The following topics describe how to publish logs using the embedded metric format, view your metrics and logs in the console, and set alarms on metrics created with the embedded metric format.

Topics

Publishing logs with the embedded metric format

You can generate embedded metric format logs using the following methods:

- Generate and send the logs by using the open-sourced client libraries.
- Manually generate the logs using the embedded metric format specification, and then use the CloudWatch agent or the PutLogEvents API to send the logs.

The following topics provide more information about embedded metrics.

Topics

Creating logs in embedded metric format using the client libraries

You can use the open-sourced client libraries that Amazon provides to create embedded metric format logs. Currently, these libraries are available for the languages in the following list. Full examples for different setups can be found in our client libraries under **/examples**. The libraries and the instructions for how to use them are located on Github. Use the following links.

- Node.Js

> **Note**
>
> For Node.js, versions 4.1.1+, 3.0.2+, 2.0.7+ are required for use with the Lambda JSON log format. Using previous versions in such Lambda environments will lead to metric loss. For more information, see Accessing Amazon CloudWatch logs for Amazon Lambda.

- Python
- Java
- C#

Client libraries are meant to work out of the box with the CloudWatch agent. Generated embedded metric format logs are sent to the CloudWatch agent, which then aggregates and publishes them to CloudWatch Logs for you.

> **Note**
>
> When using Lambda, no agent is required to send the logs to CloudWatch. Anything logged to STDOUT is sent to CloudWatch Logs via the Lambda Logging Agent.

Specification: Embedded metric format

The CloudWatch embedded metric format is a JSON specification used to instruct CloudWatch Logs to automatically extract metric values embedded in structured log events. You can use CloudWatch to graph and create alarms on the extracted metric values. This section describes embedded metric format specification conventions and the embedded metric format document structure.

Embedded metric format specification conventions

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this format specification are to be interpreted as described in Key Words RFC2119.

The terms "JSON", "JSON text", "JSON value", "member", "element", "object", "array", "number", "string", "boolean", "true", "false", and "null" in this format specification are to be interpreted as defined in JavaScript Object Notation RFC8259.

> **Note**
>
> If you plan to create alarms on metrics created using embedded metric format, see Setting alarms on metrics created with the embedded metric format for recommendations.

Embedded metric format document structure

This section describes the structure of an embedded metric format document. Embedded metric format documents are defined in JavaScript Object Notation RFC8259.

Unless otherwise noted, objects defined by this specification MUST NOT contain any additional members. Members not recognized by this specification MUST be ignored. Members defined in this specification are case-sensitive.

The embedded metric format is subject to the same limits as standard CloudWatch Logs events and are limited to a maximum size of 1 MB.

With the embedded metric format, you can track the processing of your EMF logs by metrics that are published in the `Amazon/Logs` namespace of your account. These can be used to track failed metric generation from EMF, as well as whether failures happen due to parsing or validation. For more details see Monitoring with CloudWatch metrics.

Root node

The LogEvent message MUST be a valid JSON object with no additional data at the beginning or end of the LogEvent message string. For more information about the LogEvent structure, see InputLogEvent.

Embedded metric format documents MUST contain the following top-level member on the root node. This is a Metadata object object.

```
{
 "_aws": {
    "CloudWatchMetrics": [ ... ]
  }
}
```

The root node MUST contain all Target members members defined by the references in the MetricDirective object.

The root node MAY contain any other members that are not included in the above requirements. The values of these members MUST be valid JSON types.

Metadata object

The _aws member can be used to represent metadata about the payload that informs downstream services how they should process the LogEvent. The value MUST be an object and MUST contain the following members:

- **CloudWatchMetrics**— An array of MetricDirective object used to instruct CloudWatch to extract metrics from the root node of the LogEvent.

```
{
  "_aws": {
    "CloudWatchMetrics": [ ... ]
  }
}
```

- **Timestamp**— A number representing the time stamp used for metrics extracted from the event. Values MUST be expressed as the number of milliseconds after Jan 1, 1970 00:00:00 UTC.

```
{
  "_aws": {
    "Timestamp": 1559748430481
  }
}
```

MetricDirective object

The MetricDirective object instructs downstream services that the LogEvent contains metrics that will be extracted and published to CloudWatch. MetricDirectives MUST contain the following members:

- **Namespace**— A string representing the CloudWatch namespace for the metric.
- **Dimensions**— A DimensionSet array.
- **Metrics**— An array of MetricDefinition objects. This array MUST NOT contain more than 100 MetricDefinition objects.

DimensionSet array

A DimensionSet is an array of strings containing the dimension keys that will be applied to all metrics in the document. The values within this array MUST also be members on the root-node— referred to as the Target members

A DimensionSet MUST NOT contain more than 30 dimension keys. A DimensionSet MAY be empty.

The target member MUST have a string value. This value MUST NOT contain more than 1024 characters. The target member defines a dimension that will be published as part of the metric identity. Every DimensionSet used creates a new metric in CloudWatch. For more information about dimensions, see Dimension and Dimensions.

```
{
 "_aws": {
   "CloudWatchMetrics": [
     {
       "Dimensions": [ [ "functionVersion" ] ],
       ...
     }
   ]
 },
 "functionVersion": "$LATEST"
}
```

> **Note**
>
> Be careful when configuring your metric extraction as it impacts your custom metric usage and corresponding bill. If you unintentionally create metrics based on high-cardinality dimensions (such as `requestId`), the embedded metric format will by design create a custom metric corresponding to each unique dimension combination. For more information, see Dimensions.

MetricDefinition object

A MetricDefinition is an object that MUST contain the following member:

- **Name**— A string Reference values to a metric Target members. Metric targets MUST be either a numeric value or an array of numeric values.

A MetricDefinition object MAY contain the following members:

- **Unit**— An OPTIONAL string value representing the unit of measure for the corresponding metric. Values SHOULD be valid CloudWatch metric units. For information about valid units, see MetricDatum. If a value is not provided, then a default value of NONE is assumed.
- **StorageResolution**— An OPTIONAL integer value representing the storage resolution for the corresponding metric. Setting this to 1 specifies this metric as a high-resolution metric, so that CloudWatch stores the metric with sub-minute resolution down to one second. Setting this to 60 specifies this metric as standard-resolution, which CloudWatch stores at 1-minute resolution. Values SHOULD be valid CloudWatch supported resolutions, 1 or 60. If a value is not provided, then a default value of 60 is assumed.

  For more information about high-resolution metrics, see High-resolution metrics.

> **Note**
>
> If you plan to create alarms on metrics created using embedded metric format, see Setting alarms on metrics created with the embedded metric format for recommendations.

```
{
  "_aws": {
    "CloudWatchMetrics": [
      {
        "Metrics": [
          {
            "Name": "Time",
            "Unit": "Milliseconds",
            "StorageResolution": 60
          }
        ],
        ...
      }
    ]
  },
  "Time": 1
}
```

Reference values

Reference values are string values that reference Target members members on the root node. These references should NOT be confused with the JSON Pointers described in RFC6901. Target values cannot be nested.

Target members

Valid targets MUST be members on the root node and cannot be nested objects. For example, a _reference_ value of "A.a" MUST match the following member:

```
{ "A.a" }
```

It MUST NOT match the nested member:

```
{ "A": { "a" } }
```

Valid values of target members depend on what is referencing them. A metric target MUST be a numeric value or an array of numeric values. Numeric array metric targets MUST NOT have more than 100 members. A dimension target MUST have a string value.

Embedded metric format example and JSON schema

The following is a valid example of embedded metric format.

```
{
  "_aws": {
    "Timestamp": 1574109732004,
    "CloudWatchMetrics": [
      {
        "Namespace": "lambda-function-metrics",
        "Dimensions": [["functionVersion"]],
        "Metrics": [
          {
            "Name": "time",
            "Unit": "Milliseconds",
            "StorageResolution": 60
          }
        ]
      }
    ]
```

```
  },
  "functionVersion": "$LATEST",
  "time": 100,
  "requestId": "989ffbf8-9ace-4817-a57c-e4dd734019ee"
}
```

You can use the following schema to validate embedded metric format documents.

```
{
    "type": "object",
    "title": "Root Node",
    "required": [
        "_aws"
    ],
    "properties": {
        "_aws": {
            "$id": "#/properties/_aws",
            "type": "object",
            "title": "Metadata",
            "required": [
                "Timestamp",
                "CloudWatchMetrics"
            ],
            "properties": {
                "Timestamp": {
                    "$id": "#/properties/_aws/properties/Timestamp",
                    "type": "integer",
                    "title": "The Timestamp Schema",
                    "examples": [
                        1565375354953
                    ]
                },
                "CloudWatchMetrics": {
                    "$id": "#/properties/_aws/properties/CloudWatchMetrics",
                    "type": "array",
                    "title": "MetricDirectives",
                    "items": {
                        "$id": "#/properties/_aws/properties/CloudWatchMetrics/items",
                        "type": "object",
                        "title": "MetricDirective",
                        "required": [
                            "Namespace",
                            "Dimensions",
```

```
                                "Metrics"
                            ],
                            "properties": {
                                "Namespace": {
                                    "$id": "#/properties/_aws/properties/CloudWatchMetrics/
items/properties/Namespace",
                                    "type": "string",
                                    "title": "CloudWatch Metrics Namespace",
                                    "examples": [
                                        "MyApp"
                                    ],
                                    "pattern": "^(.*)$",
                                    "minLength": 1,
                                    "maxLength": 1024
                                },
                                "Dimensions": {
                                    "$id": "#/properties/_aws/properties/CloudWatchMetrics/
items/properties/Dimensions",
                                    "type": "array",
                                    "title": "The Dimensions Schema",
                                    "minItems": 1,
                                    "items": {
                                        "$id": "#/properties/_aws/properties/
CloudWatchMetrics/items/properties/Dimensions/items",
                                        "type": "array",
                                        "title": "DimensionSet",
                                        "minItems": 0,
                                        "maxItems": 30,
                                        "items": {
                                            "$id": "#/properties/_aws/properties/
CloudWatchMetrics/items/properties/Dimensions/items/items",
                                            "type": "string",
                                            "title": "DimensionReference",
                                            "examples": [
                                                "Operation"
                                            ],
                                            "pattern": "^(.*)$",
                                            "minLength": 1,
                                            "maxLength": 250
}
                                    }
                                },
                                "Metrics": {
```

```
                                "$id": "#/properties/_aws/properties/CloudWatchMetrics/
items/properties/Metrics",
                                "type": "array",
                                "title": "MetricDefinitions",
                                "items": {
                                    "$id": "#/properties/_aws/properties/
CloudWatchMetrics/items/properties/Metrics/items",
                                    "type": "object",
                                    "title": "MetricDefinition",
                                    "required": [
                                        "Name"
                                    ],
                                    "properties": {
                                        "Name": {
                                            "$id": "#/properties/_aws/properties/
CloudWatchMetrics/items/properties/Metrics/items/properties/Name",
                                            "type": "string",
                                            "title": "MetricName",
                                            "examples": [
                                                "ProcessingLatency"
                                            ],
                                            "pattern": "^(.*)$",
                                            "minLength": 1,
                                            "maxLength": 1024
                                        },
                                        "Unit": {
                                            "$id": "#/properties/_aws/properties/
CloudWatchMetrics/items/properties/Metrics/items/properties/Unit",
                                            "type": "string",
                                            "title": "MetricUnit",
                                            "examples": [
                                                "Milliseconds"
                                            ],
                                            "pattern": "^(Seconds|Microseconds|
Milliseconds|Bytes|Kilobytes|Megabytes|Gigabytes|Terabytes|Bits|Kilobits|Megabits|
Gigabits|Terabits|Percent|Count|Bytes\\/Second|Kilobytes\\/Second|Megabytes\\/Second|
Gigabytes\\/Second|Terabytes\\/Second|Bits\\/Second|Kilobits\\/Second|Megabits\\/
Second|Gigabits\\/Second|Terabits\\/Second|Count\\/Second|None)$"
                                         },
                                         "StorageResolution": {
                                            "$id": "#/properties/_aws/properties/
CloudWatchMetrics/items/properties/Metrics/items/properties/StorageResolution",
                                            "type": "integer",
                                            "title": "StorageResolution",
```

```
                                            "examples": [
                                                60
                                            ]
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
```

Using the PutLogEvents API to send manually-created embedded metric format logs

You can send embedded metric format logs to CloudWatch Logs using the CloudWatch Logs PutLogEvents API. When calling PutLogEvents, you have the option to include the following HTTP header, which tells CloudWatch Logs the metrics should be extracted, but it's not required.

```
x-amzn-logs-format: json/emf
```

The following is a full example using the Amazon SDK for Java 2.x:

```
package org.example.basicapp;

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.cloudwatchlogs.CloudWatchLogsClient;
import software.amazon.awssdk.services.cloudwatchlogs.model.DescribeLogStreamsRequest;
import software.amazon.awssdk.services.cloudwatchlogs.model.DescribeLogStreamsResponse;
import software.amazon.awssdk.services.cloudwatchlogs.model.InputLogEvent;
import software.amazon.awssdk.services.cloudwatchlogs.model.PutLogEventsRequest;

import java.util.Collections;

public class EmbeddedMetricsExample {
        public static void main(String[] args) {

                final String usage = "To run this example, supply a Region code (eg.
 us-east-1), log group, and stream name as command line arguments"
```

```
                                + "Ex: PutLogEvents <region-id> <log-group-name>
 <stream-name>";

                if (args.length != 3) {
                        System.out.println(usage);
                        System.exit(1);
                }

                String regionId = args[0];
                String logGroupName = args[1];
                String logStreamName = args[2];

                CloudWatchLogsClient logsClient =
 CloudWatchLogsClient.builder().region(Region.of(regionId)).build();

                // Build a JSON log using the EmbeddedMetricFormat.
                long timestamp = System.currentTimeMillis();
                String message = "{" +
                                "  \"_aws\": {" +
                                "    \"Timestamp\": " + timestamp  + "," +
                                "    \"CloudWatchMetrics\": [" +
                                "      {" +
                                "        \"Namespace\": \"MyApp\"," +
                                "        \"Dimensions\": [[\"Operation\"], [\"Operation
\", \"Cell\"]]," +
                                "        \"Metrics\": [{ \"Name\": \"ProcessingLatency
\", \"Unit\": \"Milliseconds\", \"StorageResolution\": 60 }]" +
                                "      }" +
                                "    ]" +
                                "  }," +
                                "  \"Operation\": \"Aggregator\"," +
                                "  \"Cell\": \"001\"," +
                                "  \"ProcessingLatency\": 100" +
                                "}";
                InputLogEvent inputLogEvent = InputLogEvent.builder()
                        .message(message)
                        .timestamp(timestamp)
                        .build();

                // Specify the request parameters.
                PutLogEventsRequest putLogEventsRequest = PutLogEventsRequest.builder()
                        .logEvents(Collections.singletonList(inputLogEvent))
                        .logGroupName(logGroupName)
                        .logStreamName(logStreamName)
```

```
                        .build();

                logsClient.putLogEvents(putLogEventsRequest);

                System.out.println("Successfully put CloudWatch log event");
        }

}
```

> **Note**
>
> With the embedded metric format, you can track the processing of your EMF logs by metrics that are published in the `Amazon/Logs` namespace of your account. These can be used to track failed metric generation from EMF, as well as whether failures happen due to parsing or validation. For more details see Monitoring with CloudWatch metrics.

Using the CloudWatch agent to send embedded metric format logs

This section describes how to install and use the CloudWatch agent. The first part of this section describes how to install the CloudWatch agent. The the second part of this section describes how to use the CloudWatch agent to send embedded metric format logs. If you want to use this method, you must install the CloudWatch agent for the Amazon Web Services services you want to send embedded metric format logs from. Then you can begin sending the events. The CloudWatch agent must be version 1.230621.0 or later.

> **Note**
>
> You do not need to install the CloudWatch agent to send logs from Lambda functions. Lambda function timeouts are not handled automatically. This means that if your function times out before the metrics get flushed, then the metrics for that invocation will not be captured.

Installing the CloudWatch agent

Install the CloudWatch agent for each service which is to send embedded metric format logs.

Installing the CloudWatch agent on EC2

First, install the CloudWatch agent on the instance. For more information, see Install the CloudWatch agent.

Once you have installed the agent, configure the agent to listen on a UDP or TCP port for the embedded metric format logs. The following is an example of this configuration that listens on the default socket `tcp:25888`. For more information about agent configuration, see Manually create or edit the CloudWatch agent configuration file.

```
{
  "logs": {
    "metrics_collected": {
      "emf": { }
    }
  }
}
```

Installing the CloudWatch agent on Amazon ECS

The easiest way to deploy the CloudWatch agent on Amazon ECS is to run it as a sidecar, defining it in the same task definition as your application.

Create agent configuration file

Create your CloudWatch agent configuration file locally. In this example, the relative file path will be `amazon-cloudwatch-agent.json`.

For more information about agent configuration, see Manually create or edit the CloudWatch agent configuration file.

```
{
  "logs": {
    "metrics_collected": {
      "emf": { }
    }
  }
}
```

Push configuration to SSM Parameter Store

Enter the following command to push the CloudWatch agent configuration file to the Amazon Systems Manager (SSM) Parameter Store.

```
aws ssm put-parameter \
    --name "cwagentconfig" \
    --type "String" \
    --value "`cat amazon-cloudwatch-agent.json`" \
    --region "{{region}}"
```

Configure the task definition

Configure your task definition to use the CloudWatch Agent and expose the TCP or UDP port. The sample task definition that you should use depends on your networking mode.

Notice that the `webapp` specifies the `AWS_EMF_AGENT_ENDPOINT` environment variable. This is used by the library and should point to the endpoint that the agent is listening on. Additionally, the `cwagent` specifies the `CW_CONFIG_CONTENT` as a “valueFrom” parameter that points to the SSM configuration that you created in the previous step.

This section contains one example for bridge mode and one example for host or awsvpc mode. For more examples of how you can configure the CloudWatch agent on Amazon ECS, see the [Github samples repository]

The following is an example for bridge mode. When bridge mode networking is enabled, the agent needs to be linked to your application using the `links` parameter and must be addressed using the container name.

```
{
  "containerDefinitions": [
          {
              "name": "webapp",
              "links": [ "cwagent" ],
              "image": "my-org/web-app:latest",
              "memory": 256,
              "cpu": 256,
              "environment": [{
                "name": "AWS_EMF_AGENT_ENDPOINT",
                "value": "tcp://cwagent:25888"
              }],
          },
          {
```

```
                "name": "cwagent",
                "mountPoints": [],
                "image": "public.ecr.aws/cloudwatch-agent/cloudwatch-agent:latest",
                "memory": 256,
                "cpu": 256,
                "portMappings": [{
                    "protocol": "tcp",
                    "containerPort": 25888
                }],
                "environment": [{
                  "name": "CW_CONFIG_CONTENT",
                  "valueFrom": "cwagentconfig"
                }],
            }
        ],
}
```

The following is an example for host mode or awsvpc mode. When running on these network modes, the agent can be addressed over `localhost`.

```
{
  "containerDefinitions": [
            {
                "name": "webapp",
                "image": "my-org/web-app:latest",
                "memory": 256,
                "cpu": 256,
                "environment": [{
                  "name": "AWS_EMF_AGENT_ENDPOINT",
                  "value": "tcp://127.0.0.1:25888"
                }],
            },
            {
                "name": "cwagent",
                "mountPoints": [],
                "image": "public.ecr.aws/cloudwatch-agent/cloudwatch-agent:latest",
                "memory": 256,
                "cpu": 256,
                "portMappings": [{
                    "protocol": "tcp",
                    "containerPort": 25888
                }],
                "environment": [{
```

```
                "name": "CW_CONFIG_CONTENT",
                "valueFrom": "cwagentconfig"
              }],
          }
      ],
}
```

> **Note**
>
> In awsvpc mode, you must either give a public IP address to the VPC (Fargate only), set up a NAT gateway, or set up a CloudWatch Logs VPC endpoint. For more information about setting up a NAT, see [NAT Gateways](). For more information about setting up a CloudWatch Logs VPC endpoint, see [Using CloudWatch Logs with Interface VPC Endpoints]().
> The following is an example of how to assign a public IP address to a task that uses the Fargate launch type.
>
> ```
> aws ecs run-task \
> --cluster {{cluster-name}} \
> --task-definition cwagent-fargate \
> --region {{region}} \
> --launch-type FARGATE \
> --network-configuration
>  "awsvpcConfiguration={subnets=[{{subnetId}}],securityGroups=[{{sgId}}],assignPublicIp=ENA
> ```

Ensure permissions

Ensure the IAM role executing your tasks has permission to read from the SSM Parameter Store. You can add this permission by attaching the **AmazonSSMReadOnlyAccess** policy. To do so, enter the following command.

```
aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/AmazonSSMReadOnlyAccess
 \
--role-name CWAgentECSExecutionRole
```

Installing the CloudWatch agent on Amazon EKS

Parts of this process can be skipped if you have already installed CloudWatch Container Insights on this cluster.

Permissions

If you have not already installed Container Insights, then first ensure that your Amazon EKS nodes have the appropriate IAM permissions. They should have the **CloudWatchAgentServerPolicy** attached. For more information, see Verifying prerequisites for Container Insights in CloudWatch.

Create ConfigMap

Create a ConfigMap for the agent. The ConfigMap also tells the agent to listen on a TCP or UDP port. Use the following ConfigMap.

```
# cwagent-emf-configmap.yaml
apiVersion: v1
data:
  # Any changes here must not break the JSON format
  cwagentconfig.json: |
    {
      "agent": {
        "omit_hostname": true
      },
      "logs": {
        "metrics_collected": {
          "emf": { }
        }
      }
    }
kind: ConfigMap
metadata:
  name: cwagentemfconfig
  namespace: default
```

If you have already installed Container Insights, add the following `"emf": { }` line to your existing ConfigMap.

Apply the ConfigMap

Enter the following command to apply the ConfigMap.

```
kubectl apply -f cwagent-emf-configmap.yaml
```

Deploy the agent

To deploy the CloudWatch agent as a sidecar, add the agent to your pod definition, as in the following example.

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp
  namespace: default
spec:
  containers:
    # Your container definitions go here
    - name: web-app
      image: my-org/web-app:latest
    # CloudWatch Agent configuration
    - name: cloudwatch-agent
      image: public.ecr.aws/cloudwatch-agent/cloudwatch-agent:latest
      imagePullPolicy: Always
      resources:
        limits:
          cpu: 200m
          memory: 100Mi
        requests:
          cpu: 200m
          memory: 100Mi
      volumeMounts:
        - name: cwagentconfig
          mountPath: /etc/cwagentconfig
      ports:
  # this should match the port configured in the ConfigMap
        - protocol: TCP
          hostPort: 25888
          containerPort: 25888
  volumes:
    - name: cwagentconfig
      configMap:
        name: cwagentemfconfig
```

Using the CloudWatch agent to send embedded metric format logs

When you have the CloudWatch agent installed and running, you can send the embedded metric format logs over TCP or UDP. There are two requirements when sending the logs over the agent:

- The logs must contain a `LogGroupName` key that tells the agent which log group to use.

- Each log event must be on a single line. In other words, a log event cannot contain the newline (\n) character.

The log events must also follow the embedded metric format specification. For more information, see Specification: Embedded metric format.

If you plan to create alarms on metrics created using embedded metric format, see Setting alarms on metrics created with the embedded metric format for recommendations.

The following is an example of sending log events manually from a Linux bash shell. You can instead use the UDP socket interfaces provided by your programming language of choice.

```
echo '{"_aws":{"Timestamp":1574109732004,"LogGroupName":"Foo","CloudWatchMetrics":
[{"Namespace":"MyApp","Dimensions":[["Operation"]],"Metrics":
[{"Name":"ProcessingLatency","Unit":"Milliseconds","StorageResolution":60}]}]},"Operation":"Agg
 \
> /dev/udp/0.0.0.0/25888
```

> **Note**
>
> With the embedded metric format, you can track the processing of your EMF logs by metrics that are published in the `Amazon/Logs` namespace of your account. These can be used to track failed metric generation from EMF, as well as whether failures happen due to parsing or validation. For more details see Monitoring with CloudWatch metrics.

Using the embedded metric format with Amazon Distro for OpenTelemetry

OpenTelemetry is an open-source initiative that removes boundaries and restrictions between vendor-specific formats for tracing, logs, and metrics by offering a single set of specifications and APIs. For more information, see OpenTelemetry. You can use the embedded metric format as a part of the OpenTelemetry project. Using the embedded metric format with OpenTelemetry requires two components: an OpenTelemetry-compliant data source and the Amazon Distro for OpenTelemetry Collector enabled for use with CloudWatch embedded metric format logs.

We have preconfigured redistributions of the OpenTelemetry components, which Amazon maintains, to make onboarding as easy as possible. For more information about using

OpenTelemetry with the embedded metric format, in addition to other Amazon services, see Amazon Distro for OpenTelemetry. For additional information regarding language support and usage, see Amazon Observability on Github.

Viewing your metrics and logs in the console

After you generate embedded metric format logs that extract metrics, you can use the CloudWatch console to view the metrics. Embedded metrics have the dimensions that you specified when you generated the logs. Also, the embedded metrics that you generated using the client libraries have the following default dimensions:

- ServiceType
- ServiceName
- LogGroup

This section describes how to view these metrics in the CloudWatch console and query your extracted metrics using CloudWatch Logs Insights.

To view metrics that were generated from embedded metric format logs

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Metrics**.
3. Select a namespace that you specified for your embedded metrics when you generated them. If you used the client libraries to generate the metrics and did not specify a namespace, then select **aws-embedded-metrics**. This is the default namespace for embedded metrics generated using the client libraries.
4. Select a metric dimension (for example, **ServiceName**).
5. The **All metrics** tab displays all metrics for that dimension in the namespace. You can do the following:
    a. To sort the table, use the column heading.
    b. To graph a metric, select the check box next to the metric. To select all metrics, select the check box in the heading row of the table.
    c. To filter by resource, choose the resource ID and then choose **Add to search**.
    d. To filter by metric, choose the metric name and then choose **Add to search**.

Querying logs using CloudWatch Logs Insights

You can query the detailed log events associated with the extracted metrics by using CloudWatch Logs Insights to provide deep insights into the root causes of operational events. One of the benefits of extracting metrics from your logs is that you can filter your logs later by the unique metric (metric name plus unique dimension set) and metric values, to get context on the events that contributed to the aggregated metric value

For example, to get an impacted request id or x-ray trace id, you could run the following query in CloudWatch Logs Insights.

```
filter Latency > 1000 and Operation = "Aggregator"
| fields RequestId, TraceId
```

You can also perform query-time aggregation on high-cardinality keys, such as finding the customers impacted by an event. The following example illustrates this.

```
filter Latency > 1000 and Operation = "Aggregator"
| stats count() by CustomerId
```

For more information, see [Analyzing Log Data with CloudWatch Logs Insights]

Setting alarms on metrics created with the embedded metric format

In general, creating alarms on metrics generated by the embedded metric format follows the same pattern as creating alarms on any other metrics. For more information, see [Using Amazon CloudWatch alarms]. Embedded metric format metric generation depends on your log publishing flow because CloudWatch Logs needs to process logs so they can be transformed into metrics. It's important to publish logs in a timely manner, so that metric datapoints are created within the period of time that alarms are evaluated.

If you plan on using the embedded metric format to send high-resolution metrics and create alarms on these metrics, we recommended that you flush logs to CloudWatch Logs at an interval of 5 seconds or less to avoid introducing an additional delay, which can cause alarming on partial or missing data. If you are using the CloudWatch agent, you can adjust the flush interval by setting the `force_flush_interval` parameter in the CloudWatch agent configuration file. This value defaults to 5 seconds. If you are using Lambda on other platforms where you can't control the log flush interval, consider using “M out of N” alarms to control the number of datapoints used to alarm. For more information, see Evaluating an alarm.

Amazon services that publish CloudWatch metrics

You can use the following table to learn which Amazon services publish metrics to CloudWatch. For information about the metrics and dimensions, see the specified documentation.

Service	Namespace	Documentation
Amazon Amplify	AWS/AmplifyHosting	Monitoring
Amazon API Gateway	AWS/ApiGateway	Monitor API Execution with Amazon CloudWatch
Amazon AppFlow	AWS/AppFlow	Monitoring Amazon AppFlow with Amazon CloudWatch
Amazon Application Migration Service	AWS/MGN	Monitoring Application Migration Service with Amazon CloudWatch
Amazon App Runner	AWS/AppRunner	Viewing App Runner service metrics reported to CloudWatch
AppStream 2.0	AWS/AppStream	Monitoring Amazon AppStream 2.0 Resources
Amazon AppSync	AWS/AppSync	CloudWatch Metrics
Amazon Athena	AWS/Athena	Monitoring Athena Queries with CloudWatch Metrics
Amazon Aurora	AWS/RDS	Amazon Aurora metrics
Amazon Backup	AWS/Backup	Monitoring Amazon Backup Metrics with CloudWatch
Amazon Bedrock Guardrails	AWS/Bedrock/Guardrails	Monitoring Amazon Bedrock Guardrails using CloudWatch Metrics

Service	Namespace	Documentation
Amazon Billing and Cost Management	AWS/Billing	Monitoring Charges with Alerts and Notifications
Amazon Braket	AWS/Braket/By Device	Monitoring Amazon Braket with Amazon CloudWatch
Amazon Certificate Manager	AWS/CertificateManager	Supported CloudWatch Metrics
Amazon Private CA	AWS/ACMPrivateCA	Supported CloudWatch Metrics
Amazon Q Developer in chat applications	AWS/Chatbot	Monitoring Amazon Q Developer in chat applications with Amazon CloudWatch
Amazon Chime	AWS/ChimeVoiceConnector	Monitoring Amazon Chime with Amazon CloudWatch
Amazon Chime SDK	AWS/ChimeSDK	Service metrics
Amazon Client VPN	AWS/ClientVPN	Monitoring with Amazon CloudWatch
Amazon CloudFront	AWS/CloudFront	Monitoring CloudFront Activity Using CloudWatch
Amazon CloudHSM	AWS/CloudHSM	Getting CloudWatch Metrics
Amazon CloudSearch	AWS/CloudSearch	Monitoring an Amazon CloudSearch Domain with Amazon CloudWatch

Service	Namespace	Documentation
Amazon CloudTrail	AWS/CloudTrail	Supported CloudWatch metrics
CloudWatch agent	CWAgent or a custom namespace	Metrics collected by the CloudWatch agent
CloudWatch Application Signals	ApplicationSignals	Metrics collected by Application Signals
CloudWatch metric streams	AWS/CloudWatch/MetricStreams	Monitoring your metric streams with CloudWatch metrics
CloudWatch RUM	AWS/RUM	CloudWatch metrics that you can collect with CloudWatch RUM
CloudWatch Synthetics	CloudWatchSynthetics	CloudWatch metrics published by canaries
Amazon CloudWatch Logs	AWS/Logs	Monitoring Usage with CloudWatch Metrics
Amazon CodeBuild	AWS/CodeBuild	Monitoring Amazon CodeBuild
Amazon CodeGuru Reviewer		Monitoring CodeGuru Reviewer with Amazon CloudWatch
Amazon CodePipeline		Amazon CodePipeline CloudWatch metrics
Amazon Kendra		Monitoring Amazon Kendra with Amazon CloudWatch

Service	Namespace	Documentation
Amazon Cognito	AWS/Cognito	Monitoring Amazon Cognito
Amazon Comprehend	AWS/Comprehend	Monitoring Amazon Comprehend endpoints
Amazon Config	AWS/Config	Amazon Config Usage and Success Metrics
Amazon Connect	AWS/Connect	Monitoring Amazon Connect in Amazon CloudWatch Metrics
Amazon Data Lifecycle Manager	AWS/DataLifecycleManager	Monitor your policies using Amazon CloudWatch
Amazon DataSync	AWS/DataSync	Monitoring Your Task
Amazon DataZone		Monitoring Amazon DataZone with Amazon CloudWatch
Amazon DevOps Guru	AWS/DevOps-Guru	Monitoring Amazon DevOps Guru with Amazon CloudWatch
Amazon Database Migration Service	AWS/DMS	Monitoring Amazon DMS Tasks
Amazon Direct Connect	AWS/DX	Monitoring with Amazon CloudWatch
Amazon Directory Service	AWS/DirectoryService	Use Amazon CloudWatch metrics to determine when to add domain controllers
Amazon DocumentDB	AWS/DocDB	Monitoring Amazon DocumentDB with CloudWatch

Service	Namespace	Documentation
Amazon DynamoDB	AWS/DynamoDB	DynamoDB Metrics and Dimensions
DynamoDB Accelerator (DAX)	AWS/DAX	Viewing DAX Metrics and Dimensions
Amazon EC2	AWS/EC2	Monitoring Your Instances Using CloudWatch
Amazon EC2 Elastic Graphics	AWS/ElasticGPUs	Using CloudWatch metrics to monitor Elastic Graphics
Amazon EC2 Spot Fleet	AWS/EC2Spot	CloudWatch Metrics for Spot Fleet
Amazon EC2 Auto Scaling	AWS/AutoScaling	Monitoring Your Auto Scaling Groups and Instances Using CloudWatch
Amazon Elastic Beanstalk	AWS/ElasticBeanstalk	Publishing Amazon CloudWatch Custom Metrics for an Environment
Amazon Elastic Block Store	AWS/EBS	Amazon CloudWatch Metrics for Amazon EBS
Amazon Elastic Container Registry	AWS/ECR	Amazon ECR repository metrics
Amazon Elastic Container Service	AWS/ECS	Amazon ECS CloudWatch Metrics

Service	Namespace	Documentation
Amazon ECS through CloudWatch Container Insights	ECS/ContainerInsights	Amazon ECS Container Insights metrics
Amazon ECS Cluster auto scaling	AWS/ECS/ManagedScaling	Amazon ECS cluster auto scaling
Amazon Elastic Disaster Recovery		CloudWatch Metrics for DRS
Amazon Elastic File System	AWS/EFS	Monitoring with CloudWatch
Amazon Elastic Inference	AWS/ElasticInference	Using CloudWatch Metrics to Monitor Amazon Elastic Inference
Amazon EKS	AWS/EKS	Basic metrics in Amazon CloudWatch
Amazon EKS through CloudWatch Container Insights	ContainerInsights	Amazon EKS and Kubernetes Container Insights metrics
Elastic Load Balancing	AWS/ApplicationELB	CloudWatch Metrics for your Application Load Balancer
Elastic Load Balancing	AWS/NetworkELB	CloudWatch Metrics for your Network Load Balancer
Elastic Load Balancing	AWS/GatewayELB	CloudWatch Metrics for your Gateway Load Balancer

Service	Namespace	Documentation
Elastic Load Balancing	AWS/ELB	CloudWatch Metrics for your Classic Load Balancer
Amazon Elastic Transcoder	AWS/ElasticTranscoder	Monitoring with Amazon CloudWatch
Amazon ElastiCache (Memcached)	AWS/ElastiCache	Monitoring Use with CloudWatch Metrics
Amazon ElastiCache (Redis OSS)	AWS/ElastiCache	Monitoring Use with CloudWatch Metrics
Amazon OpenSearch Service	AWS/ES	Monitoring OpenSearch cluster metrics with Amazon CloudWatch
Amazon EMR	AWS/ElasticMapReduce	Monitor Metrics with CloudWatch
AWS Elemental MediaConnect	AWS/MediaConnect	Monitoring MediaConnect with Amazon CloudWatch
AWS Elemental MediaConvert	AWS/MediaConvert	Using CloudWatch Metrics to View Metrics for AWS Elemental MediaConvert Resources
AWS Elemental MediaLive	AWS/MediaLive	Monitoring activity using Amazon CloudWatch metrics
AWS Elemental MediaPackage	AWS/MediaPackage	Monitoring AWS Elemental MediaPackage with Amazon CloudWatch Metrics
AWS Elemental MediaStore	AWS/MediaStore	Monitoring AWS Elemental MediaStore with Amazon CloudWatch Metrics

Service	Namespace	Documentation
AWS Elemental MediaTailor	AWS/MediaTailor	Monitoring AWS Elemental MediaTailor with Amazon CloudWatch
Amazon End User Messaging SMS	AWS/SMSVoice	Monitoring Amazon End User Messaging SMS with Amazon CloudWatch
Amazon End User Messaging Social	AWS/SocialMessaging	Monitoring Amazon End User Messaging Social with Amazon CloudWatch
Amazon EventBridge	AWS/Events	Monitoring Amazon EventBridge
Amazon FinSpace		Logging and monitoring
Amazon Forecast		CloudWatch Metrics for Amazon Forecast
Amazon Fraud Detector		Monitoring Amazon Fraud Detector with Amazon CloudWatch
Amazon FSx for Lustre	AWS/FSx	Monitoring Amazon FSx for Lustre
Amazon FSx for OpenZFS	AWS/FSx	Monitoring with Amazon CloudWatch
Amazon FSx for Windows File Server	AWS/FSx	Monitoring Amazon FSx for Windows File Server
Amazon FSx for NetApp ONTAP	AWS/FSx	Monitoring with Amazon CloudWatch

Service	Namespace	Documentation
Amazon FSx for OpenZFS	AWS/FSx	Monitoring with Amazon CloudWatch
Amazon GameLift Servers	AWS/GameLift	Monitor Amazon GameLift Servers with CloudWatch
Amazon Global Accelerator	AWS/GlobalAccelerator	Using Amazon CloudWatch with Amazon Global Accelerator
Amazon Glue	Glue	Monitoring Amazon Glue Using CloudWatch Metrics
Amazon Ground Station	AWS/GroundStation	Metrics Using Amazon CloudWatch
Amazon HealthLake	AWS/HealthLake	Monitoring HealthLake with CloudWatch
Amazon Inspector	AWS/Inspector	Monitoring Amazon Inspector Using CloudWatch
Amazon Interactive Video Service	AWS/IVS	Monitoring Amazon IVS with Amazon CloudWatch
Amazon Interactive Video Service Chat	AWS/IVSChat	Monitoring Amazon IVS with Amazon CloudWatch
Amazon IoT	AWS/IoT	Amazon IoT Metrics and Dimensions
Amazon IoT Analytics	AWS/IoTAnalytics	Namespace, Metrics, and Dimensions
Amazon IoT FleetWise	AWS/IoTFleetWise	Monitoring Amazon IoT FleetWise with Amazon CloudWatch

Service	Namespace	Documentation
Amazon IoT SiteWise	AWS/IoTSiteWise	Monitoring Amazon IoT SiteWise with Amazon CloudWatch metrics
Amazon IoT TwinMaker	AWS/IoTTwinMaker	Monitoring Amazon IoT TwinMaker with Amazon CloudWatch metrics
Amazon Key Management Service	AWS/KMS	Monitoring with CloudWatch
Amazon Keyspaces (for Apache Cassandra)	AWS/Cassandra	Amazon Keyspaces Metrics and Dimensions
Amazon Kendra		Monitoring Amazon Kendra with Amazon CloudWatch
Amazon Managed Service for Apache Flink	AWS/KinesisAnalytics	*Managed Service for Apache Flink for SQL Applications*: Monitoring with CloudWatch *Managed Service for Apache Flink for Apache Flink*: Viewing Amazon Managed Service for Apache Flink Metrics and Dimensions
Amazon Data Firehose	AWS/Firehose	Monitoring Firehose Using CloudWatch Metrics
Amazon Kinesis Data Streams	AWS/Kinesis	Monitoring Amazon Kinesis Data Streams with Amazon CloudWatch
Amazon Kinesis Video Streams	AWS/KinesisVideo	Monitoring Kinesis Video Streams Metrics with CloudWatch
Amazon Lambda	AWS/Lambda	Amazon Lambda Metrics
Amazon Lex	AWS/Lex	Monitoring Amazon Lex with Amazon CloudWatch

Service	Namespace	Documentation
Amazon License Manager	AWSLicenseManager/licenseUsage AWS/LicenseManager/LinuxSubscriptions	Monitoring license usage with Amazon CloudWatch Usage metrics and Amazon CloudWatch alarms for Linux subscriptions
Amazon Location Service	AWS/Location	Amazon Location Service metrics exported to Amazon CloudWatch
Amazon Lookout for Equipment	AWS/lookoutequipment	Monitoring Lookout for Equipment with Amazon CloudWatch
Amazon Lookout for Metrics	AWS/LookoutMetrics	Monitoring Lookout for Metrics with Amazon CloudWatch
Amazon Lookout for Vision	AWS/LookoutVision	Monitoring Lookout for Vision with Amazon CloudWatch
Amazon Mainframe Modernization		Monitoring Amazon Mainframe Modernization with Amazon CloudWatch
Amazon Machine Learning	AWS/ML	Monitoring Amazon ML with CloudWatch Metrics
Amazon Managed Blockchain	AWS/managedblockchain	Use Hyperledger Fabric Peer Node Metrics on Amazon Managed Blockchain

Service	Namespace	Documentation
Amazon Managed Service for Prometheus	AWS/Prometheus	Amazon CloudWatch metrics
Amazon Managed Streaming for Apache Kafka	AWS/Kafka	Monitoring Amazon MSK with Amazon CloudWatch
Amazon Managed Streaming for Apache Kafka	AWS/KafkaConnect	Monitoring MSK Connect
Amazon Managed Workflows for Apache Airflow	AWS/MWAA	Container, queue, and database metrics for Amazon MWAA
Amazon MemoryDB	AWS/MemoryDB	Monitoring CloudWatch metrics
Amazon MQ	AWS/AmazonMQ	Monitoring Amazon MQ Brokers Using Amazon CloudWatch
Amazon Neptune	AWS/Neptune	Monitoring Neptune with CloudWatch
Amazon Network Firewall	AWS/NetworkFirewall	Amazon Network Firewall metrics in Amazon CloudWatch
Amazon Network Manager	AWS/NetworkManager	CloudWatch metrics for on-premises resources
Amazon HealthOmics	AWS/Omics	Monitoring Amazon HealthOmics with Amazon CloudWatch

Service	Namespace	Documentation
Amazon OpsWorks	AWS/OpsWorks	Monitoring Stacks using Amazon CloudWatch
Amazon Outposts	AWS/Outposts	CloudWatch metrics for Amazon Outposts
Amazon Panorama	AWS/PanoramaDeviceMetrics	Monitoring appliances and applications with Amazon CloudWatch
Amazon Personalize	AWS/Personalize	CloudWatch metrics for Amazon Personalize
Amazon Pinpoint	AWS/Pinpoint	View Amazon Pinpoint metrics in CloudWatch
Amazon Polly	AWS/Polly	Integrating CloudWatch with Amazon Polly
Amazon PrivateLink	AWS/PrivateLinkEndpoints	CloudWatch metrics for Amazon PrivateLink
Amazon PrivateLink	AWS/PrivateLinkServices	CloudWatch metrics for Amazon PrivateLink
Amazon Private 5G	AWS/Private5G	Amazon CloudWatch metrics
Amazon Q Apps	AWS/QApps	Monitoring Amazon Q Business and Amazon Q Apps with Amazon CloudWatch
Amazon Q Business	AWS/QBusiness	Monitoring Amazon Q Business and Amazon Q Apps with Amazon CloudWatch
Amazon Q Developer	AWS/Q	Monitoring Amazon Q Developer with Amazon CloudWatch

Service	Namespace	Documentation
Amazon QLDB	AWS/QLDB	Monitoring data in Amazon QuickSight
Amazon QuickSight	AWS/QuickSight	Monitoring with Amazon CloudWatch
Amazon Redshift	AWS/Redshift	Amazon Redshift Performance Data
Amazon Relational Database Service	AWS/RDS	Monitoring Amazon RDS metrics with Amazon CloudWatch
Amazon Rekognition	AWS/Rekognition	Monitoring Rekognition with Amazon CloudWatch
Amazon Web Services re:Post Private	AWS/rePostPrivate	Monitoring Amazon Web Services re:Post Private with Amazon CloudWatch
Amazon RoboMaker	AWS/Robomaker	Monitoring Amazon RoboMaker with Amazon CloudWatch
Amazon Route 53	AWS/Route53	Monitoring Amazon Route 53
Route 53 Application Recovery Controller	AWS/Route53RecoveryReadiness	Using Amazon CloudWatch with Application Recovery Controller
Amazon SageMaker AI	AWS/SageMaker	Monitoring SageMaker AI with CloudWatch

Service	Namespace	Documentation
Amazon SageMaker AI Model Building Pipelines	AWS/SageMaker/ModelBuildingPipeline	SageMaker AI Pipelines Metrics
Amazon Secrets Manager	AWS/SecretsManager	Monitoring Secrets Manager with Amazon CloudWatch
Amazon Security Lake	AWS/SecurityLake	CloudWatch metrics for Amazon Security Lake
Service Catalog	AWS/ServiceCatalog	Service Catalog CloudWatch Metrics
Amazon Shield Advanced	AWS/DDoSProtection	Monitoring with CloudWatch
Amazon Simple Email Service	AWS/SES	Retrieving Amazon SES Event Data from CloudWatch
Amazon SimSpace Weaver	AWS/simspaceweaver	Monitoring Amazon SimSpace Weaver with Amazon CloudWatch
Amazon Simple Notification Service	AWS/SNS	Monitoring Amazon SNS with CloudWatch
Amazon Simple Queue Service	AWS/SQS	Monitoring Amazon SQS Queues Using CloudWatch
Amazon S3	AWS/S3	Monitoring Metrics with Amazon CloudWatch
S3 Storage Lens	AWS/S3/Storage-Lens	Monitor S3 Storage Lens metrics in CloudWatch

Service	Namespace	Documentation
Amazon Simple Workflow Service	AWS/SWF	Amazon SWF Metrics for CloudWatch
Amazon Step Functions	AWS/States	Monitoring Step Functions Using CloudWatch
Amazon Storage Gateway	AWS/StorageGateway	Using Amazon CloudWatch metrics
Amazon Systems Manager Run Command	AWS/SSM-RunCommand	Monitoring Run Command Metrics Using CloudWatch
Amazon Textract	AWS/Textract	CloudWatch Metrics for Amazon Textract
Amazon Timestream	AWS/Timestream	Timestream Metrics and Dimensions
Amazon Transfer for SFTP	AWS/Transfer	Amazon SFTP CloudWatch Metrics
Amazon Transcribe	AWS/Transcribe	Monitoring Amazon Transcribe with Amazon CloudWatch
Amazon Translate	AWS/Translate	CloudWatch Metrics and Dimensions for Amazon Translate
Amazon Trusted Advisor	AWS/TrustedAdvisor	Creating Trusted Advisor Alarms Using CloudWatch
Amazon VPC	AWS/NATGateway	Monitoring Your NAT Gateway with CloudWatch
Amazon VPC	AWS/TransitGateway	CloudWatch Metrics for Your Transit Gateways

Service	Namespace	Documentation
Amazon VPC	AWS/VPN	Monitoring with CloudWatch
Amazon VPC IP Address Manager	AWS/IPAM	Create alarms with Amazon CloudWatch
Amazon WAF	AWS/WAFV2 for Amazon WAF resources WAF for Amazon WAF classic resources	Monitoring with CloudWatch
Amazon WorkMail	AWS/WorkMail	Monitoring Amazon WorkMail with Amazon CloudWatch
Amazon WorkSpaces	AWS/WorkSpaces	Monitor Your WorkSpaces Using CloudWatch Metrics
Amazon WorkSpaces Web	AWS/WorkSpacesWeb	Monitoring Amazon WorkSpaces Web with Amazon CloudWatch

Amazon usage metrics

CloudWatch collects metrics that track the usage of some Amazon resources and APIs. These metrics are published in the `AWS/Usage` namespace. Usage metrics in CloudWatch allow you to proactively manage usage by visualizing metrics in the CloudWatch console, creating custom dashboards, detecting changes in activity with CloudWatch anomaly detection, and configuring alarms that alert you when usage approaches a threshold.

Some Amazon services integrate these usage metrics with Service Quotas. For these services, you can use CloudWatch to manage your account's use of your service quotas. For more information, see Visualizing your service quotas and setting alarms.

Topics

Visualizing your service quotas and setting alarms

For some Amazon services, you can use the usage metrics to visualize your current service usage on CloudWatch graphs and dashboards. You can use a CloudWatch metric math function to display the service quotas for those resources on your graphs. You can also configure alarms that alert you when your usage approaches a service quota. For more information about service quotas, see What Is Service Quotas in the *Service Quotas User Guide*.

If you are signed in to an account that is set up as a monitoring account in CloudWatch cross-account observability, you can use that monitoring account to visualize service quotas and set alarms for metrics in the source accounts that are linked to that monitoring account. For more information, see CloudWatch cross-account observability.

Currently, the following services integrate their usage metrics with Service Quotas:

- Amazon CloudHSM
- Amazon Chime SDK
- Amazon CloudWatch

- Amazon CloudWatch Logs
- Amazon DynamoDB
- Amazon EC2
- Amazon Elastic Container Registry
- Elastic Load Balancing
- Amazon Fargate
- Amazon Fault Injection Service
- Amazon Interactive Video Service
- Amazon Key Management Service
- Amazon Data Firehose
- Amazon Location Service
- Amazon Managed Blockchain (AMB) Query
- Amazon RoboMaker
- Amazon SageMaker AI

To visualize a service quota and optionally set an alarm

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Metrics**.
3. On the **All metrics** tab, choose **Usage**, and then choose **By Amazon Resource**.

   The list of service quota usage metrics appears.
4. Select the check box next to one of the metrics.

   The graph displays your current usage of that Amazon resource.
5. To add your service quota to the graph, do the following:

   a. Choose the **Graphed metrics** tab.

   b. Choose **Math expression**, **Start with an empty expression**. In the new row, under **Details**, enter **`SERVICE_QUOTA(m1)`**.

      A new line is added to the graph, displaying the service quota for the resource represented in the metric.

6. To see your current usage as a percentage of the quota, add a new expression or change the current **SERVICE_QUOTA** expression. The new expression to use is **`m1/SERVICE_QUOTA(m1)*100`**.
7. (Optional) To set an alarm that notifies you if you approach the service quota, do the following:

   a. On the row with **`m1/SERVICE_QUOTA(m1)*100`**, under **Actions**, choose the alarm icon. It looks like a bell.

      The alarm creation page appears.
   b. Under **Conditions**, ensure that **Threshold type** is **Static** and **Whenever Expression1 is** is set to **Greater**. Under **than**, enter **`80`**. This creates an alarm that goes into ALARM state when your usage exceeds 80 percent of the quota.
   c. Choose **Next**.
   d. On the next page, select an Amazon SNS topic or create a new one, and then choose **Next**. The topic you select is notified when the alarm goes to ALARM state.
   e. On the next page, enter a name and description for the alarm, and then choose **Next**.
   f. Choose **Create alarm**.

Amazon API usage metrics

Most APIs that support Amazon CloudTrail logging also report usage metrics to CloudWatch. API usage metrics in CloudWatch allow you to proactively manage API usage by visualizing metrics in the CloudWatch console, creating custom dashboards, detecting changes in activity with CloudWatch Anomaly Detection, and configuring alarms that alert when usage approaches a threshold.

You can use the he following table to learn about services that report API usage metrics to CloudWatch. The table lists the values to use for the `Service` dimension, so you can see the usage metrics from that service. You can use the procedure in this section to view the list of a service's APIs that report usage metrics to CloudWatch.

Service	Value for the `Service` dimension
Amazon Identity and Access Management Access Analyzer	`Access Analyzer`

Service	Value for the Service dimension
Amazon Account Management	Account Management
Alexa for Business	A4B
Amazon API Gateway	API Gateway
Amazon App Mesh	App Mesh
Amazon AppConfig	AWS AppConfig
Amazon AppFlow	AppFlow
Application Auto Scaling	Application Auto Scaling
Application Discovery Service	Application Discovery Service
Amazon AppStream	AppStream
AppStream 2.0 Image Builder	Image Builder
Amazon Athena	Athena
Amazon Audit Manager	Audit Manager
Amazon Backup	Backup
Amazon Batch	Batch
Amazon Braket	Braket
Amazon Budgets	Budgets
Amazon Certificate Manager	Certificate Manager
Amazon Chime SDK	ChimeSDK
Amazon Cloud Directory	Cloud Directory
Amazon Cloud Map	Cloud Map

Service	Value for the Service dimension
Amazon CloudFormation	CloudFormation
Amazon CloudHSM	CloudHSM
Amazon CloudSearch	CloudSearch
Amazon CloudShell	CloudShell
Amazon CloudTrail	CloudTrail
Amazon CloudWatch	CloudWatch
Amazon CloudWatch Application Signals	CloudWatch Application Signals
Amazon CloudWatch Logs	Logs
Amazon CloudWatch Application Insights	CloudWatch Application Insights
Amazon CodeBuild	CodeBuild
Amazon CodeCommit	CodeCommit
Amazon CodeGuru Profiler	CodeGuru Profiler
Amazon CodePipeline	CodePipeline
AWS CodeStar	CodeStar
AWS CodeStar Notifications	CodeStar Notifications
AWS CodeStar Connections	CodeStar Connections
Amazon Cognito Identity pools	Cognito Identity Pools
Amazon Cognito Sync	Cognito Sync
Amazon Comprehend	Comprehend
Amazon Comprehend Medical	Comprehend Medical

Service	Value for the Service dimension
Amazon Compute Optimizer	ComputeOptimzier
Amazon Connect	Connect
Amazon Connect Customer Profiles	Customer Profiles
Amazon Cost and Usage Reports	Cost and Usage Report
Amazon Cost Explorer	Cost Explorer
Amazon Data Exchange	Data Exchange
Amazon Data Lifecycle Manager	Data Lifecycle Manager
Amazon Database Migration Service	Database Migration Service
Amazon DataSync	DataSync
Amazon Detective	Detective
Device Advisor	Device Advisor
Amazon Direct Connect	Direct Connect
Amazon Directory Service	Directory Service
DynamoDB Accelerator	DynamoDBAccelerator
Amazon EC2	EC2
EC2 Auto Scaling	EC2 Auto Scaling
Amazon Elastic Container Registry	ECR Public
Amazon Elastic Container Service	ECS
Amazon Elastic File System	EFS
Amazon Elastic Kubernetes Service	EKS

Service	Value for the Service dimension
Amazon Elastic Beanstalk	Elastic Beanstalk
Amazon Elastic Inference	Elastic Inference
Elastic Load Balancing	Elastic Load Balancing
Amazon EMR	EMR Containers
Amazon Firewall Manager	Firewall Manager
Amazon FSx	FSx
Amazon GameLift Servers	GameLift
Amazon Glue DataBrew	DataBrew
Amazon Managed Grafana	Grafana
Amazon IoT Greengrass	Greengrass
Amazon Ground Station	Ground Station
Amazon Health APIs And Notifications	Amazon Health APIs And Notifications
Amazon Interactive Video Service	IVS
Amazon IoT Core	IoT
Amazon IoT Events	IoT Events
Amazon IoT RoboRunner	IoT RoboRunner
Amazon IoT SiteWise	IoT Sitewise
Amazon IoT Wireless	IoT Wireless
Amazon Kendra	Kendra
Amazon Keyspaces (for Apache Cassandra)	Keyspaces

Service	Value for the Service dimension
Amazon Managed Service for Apache Flink	Kinesis Analytics
Amazon Data Firehose	Firehose
Kinesis Video Streams	Kinesis Video Streams
Amazon Key Management Service	KMS
Amazon Lambda	Lambda
Amazon Launch Wizard	Launch Wizard
Amazon Lex	Amazon Lex
Amazon Lightsail	Lightsail
Amazon Location Service	Location
Amazon Lookout for Vision	Lookout for Vision
Amazon Machine Learning	Amazon Machine Learning
Amazon Macie	Macie
Amazon Managed Blockchain (AMB) Query	Amazon Managed Blockchain Query
Amazon Managed Services	AWS Managed Services
Amazon Marketplace Commerce Analytics	Marketplace Analytics Service
AWS Elemental MediaConnect	MediaConnect
AWS Elemental MediaConvert	MediaConvert
AWS Elemental MediaLive	MediaLive
AWS Elemental MediaStore	Mediastore
AWS Elemental MediaTailor	MediaTailor

Service	Value for the `Service` dimension
Amazon Mobile Hub	`Mobile Hub`
Amazon Network Firewall	`Network Firewall`
Amazon OpsWorks	`OpsWorks`
Amazon OpsWorks for Configuration Management	`OPsWorks CM`
Amazon Outposts	`Outposts`
Amazon Organizations	`Organizations`
Amazon RDS Performance Insights	`Performance Insights`
Amazon Pinpoint	`Pinpoint`
Amazon Private Certificate Authority	`Private Certificate Authority`
Amazon Managed Service for Prometheus	`Prometheus`
Amazon Proton	`Proton`
Amazon Quantum Ledger Database (Amazon QLDB)	`QLDB`
Amazon RDS	`RDS`
Amazon Redshift	`Redshift Data API`
Amazon Rekognition	`Rekognition`
Amazon Resource Access Manager	`Resource Access Manager`
Amazon Resource Groups	`Resource Groups`
Amazon Resource Groups Tagging API	`Resource Groups Tagging API`
Amazon RoboMaker	`RoboMaker`

Service	Value for the Service dimension
Amazon Route 53 Domains	`Route 53 Domains`
Amazon Route 53 Resolver	`Route 53 Resolver`
Amazon S3	`S3`
Amazon S3 Glacier	`Amazon S3 Glacier`
Amazon SageMaker Runtime	`Sagemaker`
Savings Plans	`Savings Plans`
Amazon Secrets Manager	`Secrets Manager`
Amazon Security Hub	`Security Hub`
Amazon Server Migration Service	`Amazon Server Migration Service`
Amazon Service Catalog AppRegistry	`Service Catalog AppRegistry`
Service Quotas	`Service Quotas`
Amazon Shield	`Shield`
Amazon Signer	`Signer`
Amazon Simple Notification Service	`SNS`
Amazon Simple Email Service	`SES`
Amazon Simple Queue Service	`SQS`
Identity Store	`Identity Store`
Storage Gateway	`Storage Gateway`
Amazon Web Services Support	`Support`
Amazon Simple Workflow Service	`SWF`

Service	Value for the Service dimension
Amazon Textract	Textract
Amazon IoT Things Graph	ThingsGraph
Amazon Timestream	Timestream
Amazon Transcribe	Transcribe
Amazon Translate	Translate
Amazon Transcribe streaming transcription	Transcribe Streaming
Amazon Transfer Family	Transfer
Amazon WAF	WAF
Amazon WorkDocs	Amazon WorkDocs
Amazon WorkLink	WorkLink
Amazon WorkMail	Amazon WorkMail
Amazon WorkSpaces	Workspaces
Amazon X-Ray	X-Ray

Some services report usage metrics for additional APIs as well. To see whether an API reports usage metrics to CloudWatch, use the CloudWatch console to see the metrics reported by that service in the AWS/Usage namespace.

To see the list of a service's APIs that report usage metrics to CloudWatch

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Metrics**.
3. On the **All metrics** tab, choose **Usage**, and then choose **By Amazon Resource**.
4. In the search box near the list of metrics, enter the name of the service. The metrics are filtered by the service you entered.

CloudWatch usage metrics

CloudWatch collects metrics that track the usage of some Amazon resources. These metrics correspond to Amazon service quotas. Tracking these metrics can help you proactively manage your quotas. For more information, see Visualizing your service quotas and setting alarms. Service quota usage metrics are in the AWS/Usage namespace and are collected every minute.

The metrics that can be published in this namespace include CallCount, ResourceCount, and ThrottleCount. These metrics are published with the dimensions Resource, Service, and Type. The Resource dimension specifies the name of the API operation being tracked. For example, the CallCount metric with the dimensions "Service": "CloudWatch", "Type": "API" and "Resource": "PutMetricData" indicates the number of times the CloudWatch PutMetricData API operation has been called in your account.

The CallCount metric does not have a specified unit. The most useful statistic for the metric is SUM, which represents the total operation count for the 1-minute period.

Metrics

Metric	Description
CallCount	The number of specified operations performed in your account.

Dimensions

Dimension	Description
Service	The name of the Amazon service containing the resource. For CloudWatch usage metrics, the value for this dimension is CloudWatch .
Class	The class of resource being tracked. CloudWatch API usage metrics use this dimension with a value of None.
Type	The type of resource being tracked. Currently, when the Service dimension is CloudWatch , the only valid value for Type is API.
Resource	The name of the API operation. Valid values include the following:

Dimension	Description
	DeleteAlarms ,DeleteDashboards ,DescribeAlarmHisto ry ,DescribeAlarms ,GetDashboard ,GetMetricData , GetMetricStatistics ,ListMetrics ,PutDashboard ,and PutMetricData

Amazon CloudWatch tutorials

The following scenarios illustrate how you can use Amazon CloudWatch. In the first scenario, you use the CloudWatch console to create a billing alarm that tracks your Amazon usage and lets you know when you have exceeded a certain spending threshold. In the second scenario, you use the Amazon Command Line Interface (Amazon CLI) to publish a single metric for a hypothetical application named *GetStarted*.

Scenarios

- Monitor your estimated charges
- Publish metrics

Scenario: Monitor your estimated charges using CloudWatch

In this scenario, you create an Amazon CloudWatch alarm to monitor your estimated charges. When you enable the monitoring of estimated charges for your Amazon account, the estimated charges are calculated and sent several times daily to CloudWatch as metric data.

Billing metric data is stored in the US East (N. Virginia) Region and reflects worldwide charges. This data includes the estimated charges for every service in Amazon that you use, as well as the estimated overall total of your Amazon charges.

You can choose to receive alerts by email when charges have exceeded a certain threshold. These alerts are triggered by CloudWatch and messages are sent using Amazon Simple Notification Service (Amazon SNS).

> **Note**
>
> For information about analyzing CloudWatch charges that you have already been billed for, see Analyzing, optimizing, and reducing CloudWatch costs.

Tasks

- Step 1: Enable billing alerts
- Step 2: Create a billing alarm

- Step 3: Check the alarm status
- Step 4: Edit a billing alarm
- Step 5: Delete a billing alarm

Step 1: Enable billing alerts

Before you can create an alarm for your estimated charges, you must enable billing alerts, so that you can monitor your estimated Amazon charges and create an alarm using billing metric data. After you enable billing alerts, you cannot disable data collection, but you can delete any billing alarms that you created.

After you enable billing alerts for the first time, it takes about 15 minutes before you can view billing data and set billing alarms.

Requirements

- You must be signed in using root user credentials or as a user who has been given permission to view billing information.
- For consolidated billing accounts, billing data for each linked account can be found by logging in as the paying account. You can view billing data for total estimated charges and estimated charges by service for each linked account, in addition to the consolidated account.
- In a consolidated billing account, member linked account metrics are captured only if the payer account enables the **Receive Billing Alerts** preference. If you change which account is your management/payer account, you must enable the billing alerts in the new management/payer account.
- The account must not be part of the Amazon Partner Network (APN) because billing metrics are not published to CloudWatch for APN accounts. For more information, see Amazon Partner Network.

To enable the monitoring of estimated charges

1. Open the Amazon Billing and Cost Management console at https://console.amazonaws.cn/costmanagement/.
2. In the navigation pane, choose **Billing Preferences**.
3. By **Alert preferences** choose **Edit**.

4. Choose **Receive CloudWatch Billing Alerts**.
5. Choose **Save preferences**.

Step 2: Create a billing alarm

> **Important**
>
> Before you create a billing alarm, you must set your Region to US East (N. Virginia). Billing metric data is stored in this Region and represents worldwide charges. You also must enable billing alerts for your account or in the management/payer account (if you are using consolidated billing). For more information, see Step 1: Enable billing alerts.

In this procedure, you create an alarm that sends a notification when your estimated charges for Amazon exceed a defined threshold.

To create a billing alarm using the CloudWatch console

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the navigation pane, choose **Alarms**, and then choose **All alarms**.
3. Choose **Create alarm**.
4. Choose **Select metric**. In the **Browse** tab, choose **Billing**, and then choose **Total Estimated Charge**.

    > **Note**
    >
    > If you don't see the **Billing/Total Estimated Charge** metric, enable billing alerts, and change your Region to US East (N. Virginia). For more information, see Enabling billing alerts.

5. Select the checkbox for the **EstimatedCharges** metric, and then choose **Select metric**.
6. For **Statistic**, choose **Maximum**.
7. For **Period**, choose **6 hours**.
8. For **Threshold type**, choose **Static**.
9. For **Whenever EstimatedCharges is . . .**, choose **Greater**.

10. For **than . . .**, define the value that you want to cause your alarm to trigger. For example, **200** USD.

    The **EstimatedCharges** metric values are only in US dollars (USD), and the currency conversion is provided by Amazon Services LLC. For more information, see What is Amazon Billing?.

11. Choose **Additional Configuration** and do the following:

    - For **Datapoints to alarm**, specify **1 out of 1**.
    - For **Missing data treatment**, choose **Treat missing data as missing**.

12. Choose **Next**.

13. Under **Notification**, ensure that **In alarm** is selected. Then specify an Amazon SNS topic to be notified when your alarm is in the `ALARM` state. The Amazon SNS topic can include your email address so that you recieve email when the billing amount crosses the threshold that you specified.

    You can select an existing Amazon SNS topic, create a new Amazon SNS topic, or use a topic ARN to notify other account. If you want your alarm to send multiple notifications for the same alarm state or for different alarm states, choose **Add notification**.

14. Choose **Next**.

15. Under **Name and description**, enter a name for your alarm.

    - (Optional) Enter a description of your alarm.

16. Choose **Next**.

17. Under **Preview and create**, make sure that your configuration is correct, and then choose **Create alarm**.

Step 3: Check the alarm status

Now, check the status of the billing alarm that you just created.

To check the alarm status

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. If necessary, change the Region to US East (N. Virginia). Billing metric data is stored in this Region and reflects worldwide charges.
3. In the navigation pane, choose **Alarms**, **All alarms**.

4. Find the row in the table for your new alarm. Until the subscription is confirmed, it is shown as "Pending confirmation". After the subscription is confirmed, refresh the console to show the updated status.

Step 4: Edit a billing alarm

For example, you may want to increase the amount of money you spend with Amazon each month from $200 to $400. You can edit your existing billing alarm and increase the monetary amount that must be exceeded before the alarm is triggered.

To edit a billing alarm

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. If necessary, change the Region to US East (N. Virginia). Billing metric data is stored in this Region and reflects worldwide charges.
3. In the navigation pane, choose **Alarms**, **All alarms**.
4. Select the checkbox next to the alarm and choose **Actions**, **Edit**.
5. For **than...**, specify the new amount that must be exceeded to trigger the alarm and send an email notification.
6. Choose **Save Changes**.

Step 5: Delete a billing alarm

If you no longer need your billing alarm, you can delete it.

To delete a billing alarm

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. If necessary, change the Region to US East (N. Virginia). Billing metric data is stored in this Region and reflects worldwide charges.
3. In the navigation pane, choose **Alarms**, **All alarms**.
4. Select the checkbox next to the alarm and choose **Actions**, **Delete**.
5. When prompted for confirmation, choose **Yes, Delete**.

Scenario: Publish metrics to CloudWatch

In this scenario, you use the Amazon Command Line Interface (Amazon CLI) to publish a single metric for a hypothetical application named *GetStarted*. If you haven't already installed and configured the Amazon CLI, see Getting Set Up with the Amazon Command Line Interface in the *Amazon Command Line Interface User Guide*.

Tasks

Step 1: Define the data configuration

In this scenario, you publish data points that track the request latency for the application. Choose names for your metric and namespace that make sense to you. For this example, name the metric *RequestLatency* and place all of the data points into the *GetStarted* namespace.

You publish several data points that collectively represent three hours of latency data. The raw data comprises 15 request latency readings distributed over three hours. Each reading is in milliseconds:

- Hour one: 87, 51, 125, 235
- Hour two: 121, 113, 189, 65, 89
- Hour three: 100, 47, 133, 98, 100, 328

You can publish data to CloudWatch as single data points or as an aggregated set of data points called a *statistic set*. You can aggregate metrics to a granularity as low as one minute. You can publish the aggregated data points to CloudWatch as a set of statistics with four predefined keys: `Sum`, `Minimum`, `Maximum`, and `SampleCount`.

You publish the data points from hour one as single data points. For the data from hours two and three, you aggregate the data points and publish a statistic set for each hour. The key values are shown in the following table.

Hour	Raw data	Sum	Minimum	Maximum	SampleCount
1	87				
1	51				
1	125				
1	235				
2	121, 113, 189, 65, 89	577	65	189	5
3	100, 47, 133, 98, 100, 328	806	47	328	6

Step 2: Add metrics to CloudWatch

After you have defined your data configuration, you are ready to add data.

To publish data points to CloudWatch

1. At a command prompt, run the following put-metric-data commands to add data for the first hour. Replace the example timestamp with a timestamp that is two hours in the past, in Universal Coordinated Time (UTC).

```
aws cloudwatch put-metric-data --metric-name RequestLatency --namespace GetStarted
 \
--timestamp 2016-10-14T20:30:00Z --value 87 --unit Milliseconds
aws cloudwatch put-metric-data --metric-name RequestLatency --namespace GetStarted
 \
--timestamp 2016-10-14T20:30:00Z --value 51 --unit Milliseconds
aws cloudwatch put-metric-data --metric-name RequestLatency --namespace GetStarted
 \
--timestamp 2016-10-14T20:30:00Z --value 125 --unit Milliseconds
aws cloudwatch put-metric-data --metric-name RequestLatency --namespace GetStarted
 \
--timestamp 2016-10-14T20:30:00Z --value 235 --unit Milliseconds
```

2. Add data for the second hour, using a timestamp that is one hour later than the first hour.

```
aws cloudwatch put-metric-data --metric-name RequestLatency --namespace GetStarted
 \
--timestamp 2016-10-14T21:30:00Z --statistic-values
 Sum=577,Minimum=65,Maximum=189,SampleCount=5 --unit Milliseconds
```

3. Add data for the third hour, omitting the timestamp to default to the current time.

```
aws cloudwatch put-metric-data --metric-name RequestLatency --namespace GetStarted
 \
--statistic-values Sum=806,Minimum=47,Maximum=328,SampleCount=6 --unit Milliseconds
```

Step 3: Get statistics from CloudWatch

Now that you have published metrics to CloudWatch, you can retrieve statistics based on those metrics using the get-metric-statistics command as follows. Be sure to specify `--start-time` and `--end-time` far enough in the past to cover the earliest timestamp that you published.

```
aws cloudwatch get-metric-statistics --namespace GetStarted --metric-name
 RequestLatency --statistics Average \
--start-time 2016-10-14T00:00:00Z --end-time 2016-10-15T00:00:00Z --period 60
```

The following is example output:

```
{
 "Datapoints": [],
 "Label": "Request:Latency"
}
```

Step 4: View graphs with the console

After you have published metrics to CloudWatch, you can use the CloudWatch console to view statistical graphs.

To view graphs of your statistics on the console

1. Open the CloudWatch console at https://console.amazonaws.cn/cloudwatch/.
2. In the **Navigation** pane, choose **Metrics**.
3. On the **All metrics** tab, in the search box, type **RequestLatency** and press Enter.

4. Select the check box for the **RequestLatency** metric. A graph of the metric data is displayed in the upper pane.

For more information, see Graphing metrics.

Using CloudWatch with an Amazon SDK

Amazon software development kits (SDKs) are available for many popular programming languages. Each SDK provides an API, code examples, and documentation that make it easier for developers to build applications in their preferred language.

SDK documentation
Amazon CLI
Amazon SDK for Java
Amazon SDK for JavaScript
Amazon SDK for .NET
Amazon SDK for PHP
Amazon Tools for PowerShell
Amazon SDK for Python (Boto3)
Amazon SDK for Ruby
Amazon SDK for SAP ABAP

For examples specific to CloudWatch, see Code examples for CloudWatch using Amazon SDKs.

Code examples for CloudWatch using Amazon SDKs

The following code examples show how to use CloudWatch with an Amazon software development kit (SDK).

Basics are code examples that show you how to perform the essential operations within a service.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios.

Scenarios are code examples that show you how to accomplish specific tasks by calling multiple functions within a service or combined with other Amazon Web Services services.

For a complete list of Amazon SDK developer guides and code examples, see Using CloudWatch with an Amazon SDK. This topic also includes information about getting started and details about previous SDK versions.

Get started

Hello CloudWatch

The following code examples show how to get started using CloudWatch.

.NET

Amazon SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

```
using Amazon.CloudWatch;
using Amazon.CloudWatch.Model;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;

namespace CloudWatchActions;
```

```
public static class HelloCloudWatch
{
    static async Task Main(string[] args)
    {
        // Use the AWS .NET Core Setup package to set up dependency injection for
 the Amazon CloudWatch service.
        // Use your AWS profile name, or leave it blank to use the default
 profile.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonCloudWatch>()
            ).Build();

        // Now the client is available for injection.
        var cloudWatchClient =
 host.Services.GetRequiredService<IAmazonCloudWatch>();

        // You can use await and any of the async methods to get a response.
        var metricNamespace = "AWS/Billing";
        var response = await cloudWatchClient.ListMetricsAsync(new
 ListMetricsRequest
        {
            Namespace = metricNamespace
        });
        Console.WriteLine($"Hello Amazon CloudWatch! Following are some metrics
 available in the {metricNamespace} namespace:");
        Console.WriteLine();
        foreach (var metric in response.Metrics.Take(5))
        {
            Console.WriteLine($"\tMetric: {metric.MetricName}");
            Console.WriteLine($"\tNamespace: {metric.Namespace}");
            Console.WriteLine($"\tDimensions: {string.Join(", ",
 metric.Dimensions.Select(m => $"{m.Name}:{m.Value}"))}");
            Console.WriteLine();
        }
    }
}
```

- For API details, see ListMetrics in *Amazon SDK for .NET API Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [Amazon Code Examples Repository](https://github.com/awsdocs/aws-doc-sdk-examples/tree/main/javav2/example_code/cloudwatch#code-examples).

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.cloudwatch.CloudWatchClient;
import software.amazon.awssdk.services.cloudwatch.model.CloudWatchException;
import software.amazon.awssdk.services.cloudwatch.model.ListMetricsRequest;
import software.amazon.awssdk.services.cloudwatch.paginators.ListMetricsIterable;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class HelloService {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                  <namespace>\s

                Where:
                  namespace - The namespace to filter against (for example, AWS/
EC2).\s
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }
```

```
        String namespace = args[0];
        Region region = Region.US_EAST_1;
        CloudWatchClient cw = CloudWatchClient.builder()
                .region(region)
                .build();

        listMets(cw, namespace);
        cw.close();
    }

    public static void listMets(CloudWatchClient cw, String namespace) {
        try {
            ListMetricsRequest request = ListMetricsRequest.builder()
                    .namespace(namespace)
                    .build();

            ListMetricsIterable listRes = cw.listMetricsPaginator(request);
            listRes.stream()
                    .flatMap(r -> r.metrics().stream())
                    .forEach(metrics -> System.out.println(" Retrieved metric is:
 " + metrics.metricName()));

        } catch (CloudWatchException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- For API details, see ListMetrics in *Amazon SDK for Java 2.x API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

```
/**
Before running this Kotlin code example, set up your development environment,
including your credentials.

For more information, see the following documentation topic:
https://docs.aws.amazon.com/sdk-for-kotlin/latest/developer-guide/setup.html
 */
suspend fun main(args: Array<String>) {
    val usage = """
        Usage:
           <namespace>
        Where:
           namespace - The namespace to filter against (for example, AWS/EC2).
    """

    if (args.size != 1) {
        println(usage)
        exitProcess(0)
    }

    val namespace = args[0]
    listAllMets(namespace)
}

suspend fun listAllMets(namespaceVal: String?) {
    val request =
        ListMetricsRequest {
            namespace = namespaceVal
        }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient
            .listMetricsPaginated(request)
            .transform { it.metrics?.forEach { obj -> emit(obj) } }
            .collect { obj ->
                println("Name is ${obj.metricName}")
                println("Namespace is ${obj.namespace}")
            }
    }
}
```

    - For API details, see ListMetrics in *Amazon SDK for Kotlin API reference*.

Code examples

Basic examples for CloudWatch using Amazon SDKs

The following code examples show how to use the basics of Amazon CloudWatch with Amazon SDKs.

Examples

Hello CloudWatch

The following code examples show how to get started using CloudWatch.

.NET

Amazon SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [Amazon Code Examples Repository](https://github.com/awsdocs/aws-doc-sdk-examples/tree/main/dotnetv3/CloudWatch#code-examples).

```
using Amazon.CloudWatch;
using Amazon.CloudWatch.Model;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;

namespace CloudWatchActions;

public static class HelloCloudWatch
{
    static async Task Main(string[] args)
    {
        // Use the AWS .NET Core Setup package to set up dependency injection for
 the Amazon CloudWatch service.
        // Use your AWS profile name, or leave it blank to use the default
 profile.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonCloudWatch>()
            ).Build();

        // Now the client is available for injection.
        var cloudWatchClient =
 host.Services.GetRequiredService<IAmazonCloudWatch>();

        // You can use await and any of the async methods to get a response.
        var metricNamespace = "AWS/Billing";
        var response = await cloudWatchClient.ListMetricsAsync(new
 ListMetricsRequest
        {
            Namespace = metricNamespace
        });
```

```
        Console.WriteLine($"Hello Amazon CloudWatch! Following are some metrics
 available in the {metricNamespace} namespace:");
        Console.WriteLine();
        foreach (var metric in response.Metrics.Take(5))
        {
            Console.WriteLine($"\tMetric: {metric.MetricName}");
            Console.WriteLine($"\tNamespace: {metric.Namespace}");
            Console.WriteLine($"\tDimensions: {string.Join(", ",
 metric.Dimensions.Select(m => $"{m.Name}:{m.Value}"))}");
            Console.WriteLine();
        }
    }
}
```

- For API details, see ListMetrics in *Amazon SDK for .NET API Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.cloudwatch.CloudWatchClient;
import software.amazon.awssdk.services.cloudwatch.model.CloudWatchException;
import software.amazon.awssdk.services.cloudwatch.model.ListMetricsRequest;
import software.amazon.awssdk.services.cloudwatch.paginators.ListMetricsIterable;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
```

```
public class HelloService {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                  <namespace>\s

                Where:
                  namespace - The namespace to filter against (for example, AWS/
EC2).\s
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String namespace = args[0];
        Region region = Region.US_EAST_1;
        CloudWatchClient cw = CloudWatchClient.builder()
                .region(region)
                .build();

        listMets(cw, namespace);
        cw.close();
    }

    public static void listMets(CloudWatchClient cw, String namespace) {
        try {
            ListMetricsRequest request = ListMetricsRequest.builder()
                    .namespace(namespace)
                    .build();

            ListMetricsIterable listRes = cw.listMetricsPaginator(request);
            listRes.stream()
                    .flatMap(r -> r.metrics().stream())
                    .forEach(metrics -> System.out.println(" Retrieved metric is:
 " + metrics.metricName()));

        } catch (CloudWatchException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
```

```
}
```

- For API details, see ListMetrics in *Amazon SDK for Java 2.x API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

```
/**
Before running this Kotlin code example, set up your development environment,
including your credentials.

For more information, see the following documentation topic:
https://docs.aws.amazon.com/sdk-for-kotlin/latest/developer-guide/setup.html
 */
suspend fun main(args: Array<String>) {
    val usage = """
        Usage:
           <namespace>
        Where:
           namespace - The namespace to filter against (for example, AWS/EC2).
    """

    if (args.size != 1) {
        println(usage)
        exitProcess(0)
    }

    val namespace = args[0]
    listAllMets(namespace)
}

suspend fun listAllMets(namespaceVal: String?) {
    val request =
```

```
        ListMetricsRequest {
            namespace = namespaceVal
        }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient
            .listMetricsPaginated(request)
            .transform { it.metrics?.forEach { obj -> emit(obj) } }
            .collect { obj ->
                println("Name is ${obj.metricName}")
                println("Namespace is ${obj.namespace}")
            }
    }
}
```

- For API details, see ListMetrics in *Amazon SDK for Kotlin API reference*.

For a complete list of Amazon SDK developer guides and code examples, see Using CloudWatch with an Amazon SDK. This topic also includes information about getting started and details about previous SDK versions.

Learn core operations for CloudWatch using an Amazon SDK

The following code examples show how to:

- List CloudWatch namespaces and metrics.
- Get statistics for a metric and for estimated billing.
- Create and update a dashboard.
- Create and add data to a metric.
- Create and trigger an alarm, then view alarm history.
- Add an anomaly detector.
- Get a metric image, then clean up resources.

.NET

Amazon SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [Amazon Code Examples Repository](#).

Run an interactive scenario at a command prompt.

```
public class CloudWatchScenario
{
    /*
    Before running this .NET code example, set up your development environment,
 including your credentials.

    To enable billing metrics and statistics for this example, make sure billing
 alerts are enabled for your account:
    https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/
monitor_estimated_charges_with_cloudwatch.html#turning_on_billing_metrics

    This .NET example performs the following tasks:
        1. List and select a CloudWatch namespace.
        2. List and select a CloudWatch metric.
        3. Get statistics for a CloudWatch metric.
        4. Get estimated billing statistics for the last week.
        5. Create a new CloudWatch dashboard with two metrics.
        6. List current CloudWatch dashboards.
        7. Create a CloudWatch custom metric and add metric data.
        8. Add the custom metric to the dashboard.
        9. Create a CloudWatch alarm for the custom metric.
       10. Describe current CloudWatch alarms.
       11. Get recent data for the custom metric.
       12. Add data to the custom metric to trigger the alarm.
       13. Wait for an alarm state.
       14. Get history for the CloudWatch alarm.
       15. Add an anomaly detector.
       16. Describe current anomaly detectors.
       17. Get and display a metric image.
       18. Clean up resources.
    */
```

```
    private static ILogger logger = null!;
    private static CloudWatchWrapper _cloudWatchWrapper = null!;
    private static IConfiguration _configuration = null!;
    private static readonly List<string> _statTypes = new List<string>
{ "SampleCount", "Average", "Sum", "Minimum", "Maximum" };
    private static SingleMetricAnomalyDetector? anomalyDetector = null!;

    static async Task Main(string[] args)
    {
        // Set up dependency injection for the Amazon service.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
LogLevel.Information)
                    .AddFilter<ConsoleLoggerProvider>("Microsoft",
LogLevel.Trace))
            .ConfigureServices((_, services) =>
            services.AddAWSService<IAmazonCloudWatch>()
            .AddTransient<CloudWatchWrapper>()
        )
        .Build();

        _configuration = new ConfigurationBuilder()
            .SetBasePath(Directory.GetCurrentDirectory())
            .AddJsonFile("settings.json") // Load settings from .json file.
            .AddJsonFile("settings.local.json",
                true) // Optionally, load local settings.
            .Build();

        logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
            .CreateLogger<CloudWatchScenario>();

        _cloudWatchWrapper =
host.Services.GetRequiredService<CloudWatchWrapper>();

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Welcome to the Amazon CloudWatch example scenario.");
        Console.WriteLine(new string('-', 80));

        try
        {
            var selectedNamespace = await SelectNamespace();
```

```
            var selectedMetric = await SelectMetric(selectedNamespace);
            await GetAndDisplayMetricStatistics(selectedNamespace,
selectedMetric);
            await GetAndDisplayEstimatedBilling();
            await CreateDashboardWithMetrics();
            await ListDashboards();
            await CreateNewCustomMetric();
            await AddMetricToDashboard();
            await CreateMetricAlarm();
            await DescribeAlarms();
            await GetCustomMetricData();
            await AddMetricDataForAlarm();
            await CheckForMetricAlarm();
            await GetAlarmHistory();
            anomalyDetector = await AddAnomalyDetector();
            await DescribeAnomalyDetectors();
            await GetAndOpenMetricImage();
            await CleanupResources();
        }
        catch (Exception ex)
        {
            logger.LogError(ex, "There was a problem executing the scenario.");
            await CleanupResources();
        }

    }

    /// <summary>
    /// Select a namespace.
    /// </summary>
    /// <returns>The selected namespace.</returns>
    private static async Task<string> SelectNamespace()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"1. Select a CloudWatch Namespace from a list of
Namespaces.");
        var metrics = await _cloudWatchWrapper.ListMetrics();
        // Get a distinct list of namespaces.
        var namespaces = metrics.Select(m => m.Namespace).Distinct().ToList();
        for (int i = 0; i < namespaces.Count; i++)
        {
            Console.WriteLine($"\t{i + 1}. {namespaces[i]}");
        }
```

```
        var namespaceChoiceNumber = 0;
        while (namespaceChoiceNumber < 1 || namespaceChoiceNumber >
namespaces.Count)
        {
            Console.WriteLine(
                "Select a namespace by entering a number from the preceding
list:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out namespaceChoiceNumber);
        }

        var selectedNamespace = namespaces[namespaceChoiceNumber - 1];

        Console.WriteLine(new string('-', 80));

        return selectedNamespace;
    }

    /// <summary>
    /// Select a metric from a namespace.
    /// </summary>
    /// <param name="metricNamespace">The namespace for metrics.</param>
    /// <returns>The metric name.</returns>
    private static async Task<Metric> SelectMetric(string metricNamespace)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"2. Select a CloudWatch metric from a namespace.");

        var namespaceMetrics = await
_cloudWatchWrapper.ListMetrics(metricNamespace);

        for (int i = 0; i < namespaceMetrics.Count && i < 15; i++)
        {
            var dimensionsWithValues = namespaceMetrics[i].Dimensions
                .Where(d => !string.Equals("None", d.Value));
            Console.WriteLine($"\t{i + 1}. {namespaceMetrics[i].MetricName} " +
                              $"{string.Join(", :", dimensionsWithValues.Select(d
=> d.Value))}");
        }

        var metricChoiceNumber = 0;
        while (metricChoiceNumber < 1 || metricChoiceNumber >
namespaceMetrics.Count)
        {
```

```
            Console.WriteLine(
                "Select a metric by entering a number from the preceding list:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out metricChoiceNumber);
        }

        var selectedMetric = namespaceMetrics[metricChoiceNumber - 1];

        Console.WriteLine(new string('-', 80));

        return selectedMetric;
    }

    /// <summary>
    /// Get and display metric statistics for a specific metric.
    /// </summary>
    /// <param name="metricNamespace">The namespace for metrics.</param>
    /// <param name="metric">The CloudWatch metric.</param>
    /// <returns>Async task.</returns>
    private static async Task GetAndDisplayMetricStatistics(string
 metricNamespace, Metric metric)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"3. Get CloudWatch metric statistics for the last
 day.");

        for (int i = 0; i < _statTypes.Count; i++)
        {
            Console.WriteLine($"\t{i + 1}. {_statTypes[i]}");
        }

        var statisticChoiceNumber = 0;
        while (statisticChoiceNumber < 1 || statisticChoiceNumber >
 _statTypes.Count)
        {
            Console.WriteLine(
                "Select a metric statistic by entering a number from the
 preceding list:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out statisticChoiceNumber);
        }

        var selectedStatistic = _statTypes[statisticChoiceNumber - 1];
        var statisticsList = new List<string> { selectedStatistic };
```

```
        var metricStatistics = await
_cloudWatchWrapper.GetMetricStatistics(metricNamespace, metric.MetricName,
statisticsList, metric.Dimensions, 1, 60);

        if (!metricStatistics.Any())
        {
            Console.WriteLine($"No {selectedStatistic} statistics found for
{metric} in namespace {metricNamespace}.");
        }

        metricStatistics = metricStatistics.OrderBy(s => s.Timestamp).ToList();
        for (int i = 0; i < metricStatistics.Count && i < 10; i++)
        {
            var metricStat = metricStatistics[i];
            var statValue =
metricStat.GetType().GetProperty(selectedStatistic)!.GetValue(metricStat, null);
            Console.WriteLine($"\t{i + 1}. Timestamp
{metricStatistics[i].Timestamp:G} {selectedStatistic}: {statValue}");
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Get and display estimated billing statistics.
    /// </summary>
    /// <param name="metricNamespace">The namespace for metrics.</param>
    /// <param name="metric">The CloudWatch metric.</param>
    /// <returns>Async task.</returns>
    private static async Task GetAndDisplayEstimatedBilling()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"4. Get CloudWatch estimated billing for the last
week.");

        var billingStatistics = await SetupBillingStatistics();

        for (int i = 0; i < billingStatistics.Count; i++)
        {
            Console.WriteLine($"\t{i + 1}. Timestamp
{billingStatistics[i].Timestamp:G} : {billingStatistics[i].Maximum}");
        }
```

```
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Get billing statistics using a call to a wrapper class.
    /// </summary>
    /// <returns>A collection of billing statistics.</returns>
    private static async Task<List<Datapoint>> SetupBillingStatistics()
    {
        // Make a request for EstimatedCharges with a period of one day for the
 past seven days.
        var billingStatistics = await _cloudWatchWrapper.GetMetricStatistics(
            "AWS/Billing",
            "EstimatedCharges",
            new List<string>() { "Maximum" },
            new List<Dimension>() { new Dimension { Name = "Currency", Value =
 "USD" } },
            7,
            86400);

        billingStatistics = billingStatistics.OrderBy(n => n.Timestamp).ToList();

        return billingStatistics;
    }

    /// <summary>
    /// Create a dashboard with metrics.
    /// </summary>
    /// <param name="metricNamespace">The namespace for metrics.</param>
    /// <param name="metric">The CloudWatch metric.</param>
    /// <returns>Async task.</returns>
    private static async Task CreateDashboardWithMetrics()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"5. Create a new CloudWatch dashboard with metrics.");
        var dashboardName = _configuration["dashboardName"];
        var newDashboard = new DashboardModel();
        _configuration.GetSection("dashboardExampleBody").Bind(newDashboard);
        var newDashboardString = JsonSerializer.Serialize(
            newDashboard,
            new JsonSerializerOptions
            {
                DefaultIgnoreCondition = JsonIgnoreCondition.WhenWritingNull
            });
```

```
        var validationMessages =
            await _cloudWatchWrapper.PutDashboard(dashboardName,
newDashboardString);

        Console.WriteLine(validationMessages.Any() ? $"\tValidation messages:" :
null);
        for (int i = 0; i < validationMessages.Count; i++)
        {
            Console.WriteLine($"\t{i + 1}. {validationMessages[i].Message}");
        }
        Console.WriteLine($"\tDashboard {dashboardName} was created.");
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// List dashboards.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task ListDashboards()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"6. List the CloudWatch dashboards in the current
account.");

        var dashboards = await _cloudWatchWrapper.ListDashboards();

        for (int i = 0; i < dashboards.Count; i++)
        {
            Console.WriteLine($"\t{i + 1}. {dashboards[i].DashboardName}");
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Create and add data for a new custom metric.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task CreateNewCustomMetric()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"7. Create and add data for a new custom metric.");

        var customMetricNamespace = _configuration["customMetricNamespace"];
```

```
        var customMetricName = _configuration["customMetricName"];

        var customData = await PutRandomMetricData(customMetricName,
 customMetricNamespace);

        var valuesString = string.Join(',', customData.Select(d => d.Value));
        Console.WriteLine($"\tAdded metric values for for metric
 {customMetricName}: \n\t{valuesString}");

        Console.WriteLine(new string('-', 80));
    }



    /// <summary>
    /// Add some metric data using a call to a wrapper class.
    /// </summary>
    /// <param name="customMetricName">The metric name.</param>
    /// <param name="customMetricNamespace">The metric namespace.</param>
    /// <returns></returns>
    private static async Task<List<MetricDatum>> PutRandomMetricData(string
 customMetricName,
        string customMetricNamespace)
    {
        List<MetricDatum> customData = new List<MetricDatum>();
        Random rnd = new Random();

        // Add 10 random values up to 100, starting with a timestamp 15 minutes
 in the past.
        var utcNowMinus15 = DateTime.UtcNow.AddMinutes(-15);
        for (int i = 0; i < 10; i++)
        {
            var metricValue = rnd.Next(0, 100);
            customData.Add(
                new MetricDatum
                {
                    MetricName = customMetricName,
                    Value = metricValue,
                    TimestampUtc = utcNowMinus15.AddMinutes(i)
                }
            );
        }

        await _cloudWatchWrapper.PutMetricData(customMetricNamespace,
 customData);
```

```
        return customData;
    }

    /// <summary>
    /// Add the custom metric to the dashboard.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task AddMetricToDashboard()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"8. Add the new custom metric to the dashboard.");

        var dashboardName = _configuration["dashboardName"];

        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];

        var validationMessages = await SetupDashboard(customMetricNamespace,
 customMetricName, dashboardName);

        Console.WriteLine(validationMessages.Any() ? $"\tValidation messages:" :
 null);
        for (int i = 0; i < validationMessages.Count; i++)
        {
            Console.WriteLine($"\t{i + 1}. {validationMessages[i].Message}");
        }
        Console.WriteLine($"\tDashboard {dashboardName} updated with metric
 {customMetricName}.");
        Console.WriteLine(new string('-', 80));
    }


    /// <summary>
    /// Set up a dashboard using a call to the wrapper class.
    /// </summary>
    /// <param name="customMetricNamespace">The metric namespace.</param>
    /// <param name="customMetricName">The metric name.</param>
    /// <param name="dashboardName">The name of the dashboard.</param>
    /// <returns>A list of validation messages.</returns>
    private static async Task<List<DashboardValidationMessage>> SetupDashboard(
        string customMetricNamespace, string customMetricName, string
 dashboardName)
    {
        // Get the dashboard model from configuration.
```

```
        var newDashboard = new DashboardModel();
        _configuration.GetSection("dashboardExampleBody").Bind(newDashboard);

        // Add a new metric to the dashboard.
        newDashboard.Widgets.Add(new Widget
        {
            Height = 8,
            Width = 8,
            Y = 8,
            X = 0,
            Type = "metric",
            Properties = new Properties
            {
                Metrics = new List<List<object>>
                    { new() { customMetricNamespace, customMetricName } },
                View = "timeSeries",
                Region = "us-east-1",
                Stat = "Sum",
                Period = 86400,
                YAxis = new YAxis { Left = new Left { Min = 0, Max = 100 } },
                Title = "Custom Metric Widget",
                LiveData = true,
                Sparkline = true,
                Trend = true,
                Stacked = false,
                SetPeriodToTimeRange = false
            }
        });

        var newDashboardString = JsonSerializer.Serialize(newDashboard,
            new JsonSerializerOptions
            { DefaultIgnoreCondition = JsonIgnoreCondition.WhenWritingNull });
        var validationMessages =
            await _cloudWatchWrapper.PutDashboard(dashboardName,
 newDashboardString);

        return validationMessages;
    }

    /// <summary>
    /// Create a CloudWatch alarm for the new metric.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task CreateMetricAlarm()
```

```
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"9. Create a CloudWatch alarm for the new metric.");

        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];

        var alarmName = _configuration["exampleAlarmName"];
        var accountId = _configuration["accountId"];
        var region = _configuration["region"];
        var emailTopic = _configuration["emailTopic"];
        var alarmActions = new List<string>();

        if (GetYesNoResponse(
                $"\tAdd an email action for topic {emailTopic} to alarm
 {alarmName}? (y/n)"))
        {
            _cloudWatchWrapper.AddEmailAlarmAction(accountId, region, emailTopic,
 alarmActions);
        }

        await _cloudWatchWrapper.PutMetricEmailAlarm(
            "Example metric alarm",
            alarmName,
            ComparisonOperator.GreaterThanOrEqualToThreshold,
            customMetricName,
            customMetricNamespace,
            100,
            alarmActions);

        Console.WriteLine($"\tAlarm {alarmName} added for metric
 {customMetricName}.");
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Describe Alarms.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task DescribeAlarms()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"10. Describe CloudWatch alarms in the current
 account.");
```

```
        var alarms = await _cloudWatchWrapper.DescribeAlarms();
        alarms = alarms.OrderByDescending(a => a.StateUpdatedTimestamp).ToList();

        for (int i = 0; i < alarms.Count && i < 10; i++)
        {
            var alarm = alarms[i];
            Console.WriteLine($"\t{i + 1}. {alarm.AlarmName}");
            Console.WriteLine($"\tState: {alarm.StateValue} for
 {alarm.MetricName} {alarm.ComparisonOperator} {alarm.Threshold}");
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Get the recent data for the metric.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task GetCustomMetricData()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"11. Get current data for new custom metric.");

        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];
        var accountId = _configuration["accountId"];

        var query = new List<MetricDataQuery>
        {
            new MetricDataQuery
            {
                AccountId = accountId,
                Id = "m1",
                Label = "Custom Metric Data",
                MetricStat = new MetricStat
                {
                    Metric = new Metric
                    {
                        MetricName = customMetricName,
                        Namespace = customMetricNamespace,
                    },
                    Period = 1,
                    Stat = "Maximum"
```

```
                }
            }
        };

        var metricData = await _cloudWatchWrapper.GetMetricData(
            20,
            true,
            DateTime.UtcNow.AddMinutes(1),
            20,
            query);

        for (int i = 0; i < metricData.Count; i++)
        {
            for (int j = 0; j < metricData[i].Values.Count; j++)
            {
                Console.WriteLine(
                    $"\tTimestamp {metricData[i].Timestamps[j]:G} Value:
 {metricData[i].Values[j]}");
            }
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Add metric data to trigger an alarm.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task AddMetricDataForAlarm()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"12. Add metric data to the custom metric to trigger
 an alarm.");

        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];
        var nowUtc = DateTime.UtcNow;
        List<MetricDatum> customData = new List<MetricDatum>
        {
            new MetricDatum
            {
                MetricName = customMetricName,
                Value = 101,
                TimestampUtc = nowUtc.AddMinutes(-2)
```

```
            },
            new MetricDatum
            {
                MetricName = customMetricName,
                Value = 101,
                TimestampUtc = nowUtc.AddMinutes(-1)
            },
            new MetricDatum
            {
                MetricName = customMetricName,
                Value = 101,
                TimestampUtc = nowUtc
            }
        };
        var valuesString = string.Join(',', customData.Select(d => d.Value));
        Console.WriteLine($"\tAdded metric values for for metric
{customMetricName}: \n\t{valuesString}");
        await _cloudWatchWrapper.PutMetricData(customMetricNamespace,
customData);

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Check for a metric alarm using the DescribeAlarmsForMetric action.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task CheckForMetricAlarm()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"13. Checking for an alarm state.");

        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];
        var hasAlarm = false;
        var retries = 10;
        while (!hasAlarm && retries > 0)
        {
            var alarms = await
_cloudWatchWrapper.DescribeAlarmsForMetric(customMetricNamespace,
customMetricName);
            hasAlarm = alarms.Any(a => a.StateValue == StateValue.ALARM);
            retries--;
            Thread.Sleep(20000);
```

```
        }

        Console.WriteLine(hasAlarm
            ? $"\tAlarm state found for {customMetricName}."
            : $"\tNo Alarm state found for {customMetricName} after 10
retries.");

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Get history for an alarm.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task GetAlarmHistory()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"14. Get alarm history.");

        var exampleAlarmName = _configuration["exampleAlarmName"];

        var alarmHistory = await
_cloudWatchWrapper.DescribeAlarmHistory(exampleAlarmName, 2);

        for (int i = 0; i < alarmHistory.Count; i++)
        {
            var history = alarmHistory[i];
            Console.WriteLine($"\t{i + 1}. {history.HistorySummary}, time
{history.Timestamp:g}");
        }
        if (!alarmHistory.Any())
        {
            Console.WriteLine($"\tNo alarm history data found for
{exampleAlarmName}.");
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Add an anomaly detector.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task<SingleMetricAnomalyDetector> AddAnomalyDetector()
```

```
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"15. Add an anomaly detector.");

        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];

        var detector = new SingleMetricAnomalyDetector
        {
            MetricName = customMetricName,
            Namespace = customMetricNamespace,
            Stat = "Maximum"
        };
        await _cloudWatchWrapper.PutAnomalyDetector(detector);
        Console.WriteLine($"\tAdded anomaly detector for metric
 {customMetricName}.");

        Console.WriteLine(new string('-', 80));
        return detector;
    }

    /// <summary>
    /// Describe anomaly detectors.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task DescribeAnomalyDetectors()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"16. Describe anomaly detectors in the current
 account.");

        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];

        var detectors = await
 _cloudWatchWrapper.DescribeAnomalyDetectors(customMetricNamespace,
 customMetricName);

        for (int i = 0; i < detectors.Count; i++)
        {
            var detector = detectors[i];
            Console.WriteLine($"\t{i + 1}.
 {detector.SingleMetricAnomalyDetector.MetricName}, state
 {detector.StateValue}");
```

```
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Fetch and open a metrics image for a CloudWatch metric and namespace.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task GetAndOpenMetricImage()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("17. Get a metric image from CloudWatch.");

        Console.WriteLine($"\tGetting Image data for custom metric.");
        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];

        var memoryStream = await
 _cloudWatchWrapper.GetTimeSeriesMetricImage(customMetricNamespace,
 customMetricName, "Maximum", 10);
        var file = _cloudWatchWrapper.SaveMetricImage(memoryStream,
 "MetricImages");

        ProcessStartInfo info = new ProcessStartInfo();

        Console.WriteLine($"\tFile saved as {Path.GetFileName(file)}.");
        Console.WriteLine($"\tPress enter to open the image.");
        Console.ReadLine();
        info.FileName = Path.Combine("ms-photos://", file);
        info.UseShellExecute = true;
        info.CreateNoWindow = true;
        info.Verb = string.Empty;

        Process.Start(info);

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Clean up created resources.
    /// </summary>
    /// <param name="metricNamespace">The namespace for metrics.</param>
    /// <param name="metric">The CloudWatch metric.</param>
```

```
    /// <returns>Async task.</returns>
    private static async Task CleanupResources()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"18. Clean up resources.");

        var dashboardName = _configuration["dashboardName"];
        if (GetYesNoResponse($"\tDelete dashboard {dashboardName}? (y/n)"))
        {
            Console.WriteLine($"\tDeleting dashboard.");
            var dashboardList = new List<string> { dashboardName };
            await _cloudWatchWrapper.DeleteDashboards(dashboardList);
        }

        var alarmName = _configuration["exampleAlarmName"];
        if (GetYesNoResponse($"\tDelete alarm {alarmName}? (y/n)"))
        {
            Console.WriteLine($"\tCleaning up alarms.");
            var alarms = new List<string> { alarmName };
            await _cloudWatchWrapper.DeleteAlarms(alarms);
        }

        if (GetYesNoResponse($"\tDelete anomaly detector? (y/n)") &&
 anomalyDetector != null)
        {
            Console.WriteLine($"\tCleaning up anomaly detector.");

            await _cloudWatchWrapper.DeleteAnomalyDetector(
                anomalyDetector);
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Get a yes or no response from the user.
    /// </summary>
    /// <param name="question">The question string to print on the console.</
param>
    /// <returns>True if the user responds with a yes.</returns>
    private static bool GetYesNoResponse(string question)
    {
        Console.WriteLine(question);
        var ynResponse = Console.ReadLine();
```

```
        var response = ynResponse != null &&
                       ynResponse.Equals("y",
                           StringComparison.InvariantCultureIgnoreCase);
        return response;
    }
}
```

Wrapper methods used by the scenario for CloudWatch actions.

```
/// <summary>
/// Wrapper class for Amazon CloudWatch methods.
/// </summary>
public class CloudWatchWrapper
{
    private readonly IAmazonCloudWatch _amazonCloudWatch;
    private readonly ILogger<CloudWatchWrapper> _logger;

    /// <summary>
    /// Constructor for the CloudWatch wrapper.
    /// </summary>
    /// <param name="amazonCloudWatch">The injected CloudWatch client.</param>
    /// <param name="logger">The injected logger for the wrapper.</param>
    public CloudWatchWrapper(IAmazonCloudWatch amazonCloudWatch,
 ILogger<CloudWatchWrapper> logger)

    {
        _logger = logger;
        _amazonCloudWatch = amazonCloudWatch;
    }

    /// <summary>
    /// List metrics available, optionally within a namespace.
    /// </summary>
    /// <param name="metricNamespace">Optional CloudWatch namespace to use when
 listing metrics.</param>
    /// <param name="filter">Optional dimension filter.</param>
    /// <param name="metricName">Optional metric name filter.</param>
    /// <returns>The list of metrics.</returns>
    public async Task<List<Metric>> ListMetrics(string? metricNamespace = null,
 DimensionFilter? filter = null, string? metricName = null)
    {
        var results = new List<Metric>();
```

```
        var paginateMetrics = _amazonCloudWatch.Paginators.ListMetrics(
            new ListMetricsRequest
            {
                Namespace = metricNamespace,
                Dimensions = filter != null ? new List<DimensionFilter>
{ filter } : null,
                MetricName = metricName
            });
        // Get the entire list using the paginator.
        await foreach (var metric in paginateMetrics.Metrics)
        {
            results.Add(metric);
        }

        return results;
    }

    /// <summary>
    /// Wrapper to get statistics for a specific CloudWatch metric.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metricName">The name of the metric.</param>
    /// <param name="statistics">The list of statistics to include.</param>
    /// <param name="dimensions">The list of dimensions to include.</param>
    /// <param name="days">The number of days in the past to include.</param>
    /// <param name="period">The period for the data.</param>
    /// <returns>A list of DataPoint objects for the statistics.</returns>
    public async Task<List<Datapoint>> GetMetricStatistics(string
metricNamespace,
        string metricName, List<string> statistics, List<Dimension> dimensions,
int days, int period)
    {
        var metricStatistics = await _amazonCloudWatch.GetMetricStatisticsAsync(
            new GetMetricStatisticsRequest()
            {
                Namespace = metricNamespace,
                MetricName = metricName,
                Dimensions = dimensions,
                Statistics = statistics,
                StartTimeUtc = DateTime.UtcNow.AddDays(-days),
                EndTimeUtc = DateTime.UtcNow,
                Period = period
            });
```

```
        return metricStatistics.Datapoints;
    }

    /// <summary>
    /// Wrapper to create or add to a dashboard with metrics.
    /// </summary>
    /// <param name="dashboardName">The name for the dashboard.</param>
    /// <param name="dashboardBody">The metric data in JSON for the dashboard.</
param>
    /// <returns>A list of validation messages for the dashboard.</returns>
    public async Task<List<DashboardValidationMessage>> PutDashboard(string
 dashboardName,
        string dashboardBody)
    {
        // Updating a dashboard replaces all contents.
        // Best practice is to include a text widget indicating this dashboard
 was created programmatically.
        var dashboardResponse = await _amazonCloudWatch.PutDashboardAsync(
            new PutDashboardRequest()
            {
                DashboardName = dashboardName,
                DashboardBody = dashboardBody
            });

        return dashboardResponse.DashboardValidationMessages;
    }


    /// <summary>
    /// Get information on a dashboard.
    /// </summary>
    /// <param name="dashboardName">The name of the dashboard.</param>
    /// <returns>A JSON object with dashboard information.</returns>
    public async Task<string> GetDashboard(string dashboardName)
    {
        var dashboardResponse = await _amazonCloudWatch.GetDashboardAsync(
            new GetDashboardRequest()
            {
                DashboardName = dashboardName
            });

        return dashboardResponse.DashboardBody;
    }
```

```
    /// <summary>
    /// Get a list of dashboards.
    /// </summary>
    /// <returns>A list of DashboardEntry objects.</returns>
    public async Task<List<DashboardEntry>> ListDashboards()
    {
        var results = new List<DashboardEntry>();
        var paginateDashboards = _amazonCloudWatch.Paginators.ListDashboards(
            new ListDashboardsRequest());
        // Get the entire list using the paginator.
        await foreach (var data in paginateDashboards.DashboardEntries)
        {
            results.Add(data);
        }

        return results;
    }

    /// <summary>
    /// Wrapper to add metric data to a CloudWatch metric.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metricData">A data object for the metric data.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> PutMetricData(string metricNamespace,
        List<MetricDatum> metricData)
    {
        var putDataResponse = await _amazonCloudWatch.PutMetricDataAsync(
            new PutMetricDataRequest()
            {
                MetricData = metricData,
                Namespace = metricNamespace,
            });

        return putDataResponse.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Get an image for a metric graphed over time.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metric">The name of the metric.</param>
    /// <param name="stat">The name of the stat to chart.</param>
```

```
    /// <param name="period">The period to use for the chart.</param>
    /// <returns>A memory stream for the chart image.</returns>
    public async Task<MemoryStream> GetTimeSeriesMetricImage(string
 metricNamespace, string metric, string stat, int period)
    {
        var metricImageWidget = new
        {
            title = "Example Metric Graph",
            view = "timeSeries",
            stacked = false,
            period = period,
            width = 1400,
            height = 600,
            metrics = new List<List<object>>
                { new() { metricNamespace, metric, new { stat } } }
        };

        var metricImageWidgetString =
 JsonSerializer.Serialize(metricImageWidget);
        var imageResponse = await _amazonCloudWatch.GetMetricWidgetImageAsync(
            new GetMetricWidgetImageRequest()
            {
                MetricWidget = metricImageWidgetString
            });

        return imageResponse.MetricWidgetImage;
    }

    /// <summary>
    /// Save a metric image to a file.
    /// </summary>
    /// <param name="memoryStream">The MemoryStream for the metric image.</param>
    /// <param name="metricName">The name of the metric.</param>
    /// <returns>The path to the file.</returns>
    public string SaveMetricImage(MemoryStream memoryStream, string metricName)
    {
        var metricFileName = $"{metricName}_{DateTime.Now.Ticks}.png";
        using var sr = new StreamReader(memoryStream);
        // Writes the memory stream to a file.
        File.WriteAllBytes(metricFileName, memoryStream.ToArray());
        var filePath = Path.Join(AppDomain.CurrentDomain.BaseDirectory,
            metricFileName);
        return filePath;
    }
```

```
    /// <summary>
    /// Get data for CloudWatch metrics.
    /// </summary>
    /// <param name="minutesOfData">The number of minutes of data to include.</
param>
    /// <param name="useDescendingTime">True to return the data descending by
 time.</param>
    /// <param name="endDateUtc">The end date for the data, in UTC.</param>
    /// <param name="maxDataPoints">The maximum data points to include.</param>
    /// <param name="dataQueries">Optional data queries to include.</param>
    /// <returns>A list of the requested metric data.</returns>
    public async Task<List<MetricDataResult>> GetMetricData(int minutesOfData,
 bool useDescendingTime, DateTime? endDateUtc = null,
        int maxDataPoints = 0, List<MetricDataQuery>? dataQueries = null)
    {
        var metricData = new List<MetricDataResult>();
        // If no end time is provided, use the current time for the end time.
        endDateUtc ??= DateTime.UtcNow;
        var timeZoneOffset =
 TimeZoneInfo.Local.GetUtcOffset(endDateUtc.Value.ToLocalTime());
        var startTimeUtc = endDateUtc.Value.AddMinutes(-minutesOfData);
        // The timezone string should be in the format +0000, so use the timezone
 offset to format it correctly.
        var timeZoneString = $"{timeZoneOffset.Hours:D2}
{timeZoneOffset.Minutes:D2}";
        var paginatedMetricData = _amazonCloudWatch.Paginators.GetMetricData(
            new GetMetricDataRequest()
            {
                StartTimeUtc = startTimeUtc,
                EndTimeUtc = endDateUtc.Value,
                LabelOptions = new LabelOptions { Timezone = timeZoneString },
                ScanBy = useDescendingTime ? ScanBy.TimestampDescending :
 ScanBy.TimestampAscending,
                MaxDatapoints = maxDataPoints,
                MetricDataQueries = dataQueries,
            });

        await foreach (var data in paginatedMetricData.MetricDataResults)
        {
            metricData.Add(data);
        }
        return metricData;
    }
```

```
    /// <summary>
    /// Add a metric alarm to send an email when the metric passes a threshold.
    /// </summary>
    /// <param name="alarmDescription">A description of the alarm.</param>
    /// <param name="alarmName">The name for the alarm.</param>
    /// <param name="comparison">The type of comparison to use.</param>
    /// <param name="metricName">The name of the metric for the alarm.</param>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="threshold">The threshold value for the alarm.</param>
    /// <param name="alarmActions">Optional actions to execute when in an alarm
 state.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> PutMetricEmailAlarm(string alarmDescription, string
 alarmName, ComparisonOperator comparison,
        string metricName, string metricNamespace, double threshold, List<string>
 alarmActions = null!)
    {
        try
        {
            var putEmailAlarmResponse = await
 _amazonCloudWatch.PutMetricAlarmAsync(
                new PutMetricAlarmRequest()
                {
                    AlarmActions = alarmActions,
                    AlarmDescription = alarmDescription,
                    AlarmName = alarmName,
                    ComparisonOperator = comparison,
                    Threshold = threshold,
                    Namespace = metricNamespace,
                    MetricName = metricName,
                    EvaluationPeriods = 1,
                    Period = 10,
                    Statistic = new Statistic("Maximum"),
                    DatapointsToAlarm = 1,
                    TreatMissingData = "ignore"
                });
            return putEmailAlarmResponse.HttpStatusCode == HttpStatusCode.OK;
        }
        catch (LimitExceededException lex)
        {
            _logger.LogError(lex, $"Unable to add alarm {alarmName}. Alarm quota
 has already been reached.");
        }
```

```
        return false;
    }

    /// <summary>
    /// Add specific email actions to a list of action strings for a CloudWatch
 alarm.
    /// </summary>
    /// <param name="accountId">The AccountId for the alarm.</param>
    /// <param name="region">The region for the alarm.</param>
    /// <param name="emailTopicName">An Amazon Simple Notification Service (SNS)
 topic for the alarm email.</param>
    /// <param name="alarmActions">Optional list of existing alarm actions to
 append to.</param>
    /// <returns>A list of string actions for an alarm.</returns>
    public List<string> AddEmailAlarmAction(string accountId, string region,
        string emailTopicName, List<string>? alarmActions = null)
    {
        alarmActions ??= new List<string>();
        var snsAlarmAction = $"arn:aws:sns:{region}:{accountId}:
{emailTopicName}";
        alarmActions.Add(snsAlarmAction);
        return alarmActions;
    }

    /// <summary>
    /// Describe the current alarms, optionally filtered by state.
    /// </summary>
    /// <param name="stateValue">Optional filter for alarm state.</param>
    /// <returns>The list of alarm data.</returns>
    public async Task<List<MetricAlarm>> DescribeAlarms(StateValue? stateValue =
 null)
    {
        List<MetricAlarm> alarms = new List<MetricAlarm>();
        var paginatedDescribeAlarms =
 _amazonCloudWatch.Paginators.DescribeAlarms(
            new DescribeAlarmsRequest()
            {
                StateValue = stateValue
            });

        await foreach (var data in paginatedDescribeAlarms.MetricAlarms)
        {
            alarms.Add(data);
```

```
        }
        return alarms;
    }

    /// <summary>
    /// Describe the current alarms for a specific metric.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metricName">The name of the metric.</param>
    /// <returns>The list of alarm data.</returns>
    public async Task<List<MetricAlarm>> DescribeAlarmsForMetric(string
 metricNamespace, string metricName)
    {
        var alarmsResult = await _amazonCloudWatch.DescribeAlarmsForMetricAsync(
            new DescribeAlarmsForMetricRequest()
            {
                Namespace = metricNamespace,
                MetricName = metricName
            });

        return alarmsResult.MetricAlarms;
    }

    /// <summary>
    /// Describe the history of an alarm for a number of days in the past.
    /// </summary>
    /// <param name="alarmName">The name of the alarm.</param>
    /// <param name="historyDays">The number of days in the past.</param>
    /// <returns>The list of alarm history data.</returns>
    public async Task<List<AlarmHistoryItem>> DescribeAlarmHistory(string
 alarmName, int historyDays)
    {
        List<AlarmHistoryItem> alarmHistory = new List<AlarmHistoryItem>();
        var paginatedAlarmHistory =
 _amazonCloudWatch.Paginators.DescribeAlarmHistory(
            new DescribeAlarmHistoryRequest()
            {
                AlarmName = alarmName,
                EndDateUtc = DateTime.UtcNow,
                HistoryItemType = HistoryItemType.StateUpdate,
                StartDateUtc = DateTime.UtcNow.AddDays(-historyDays)
            });

        await foreach (var data in paginatedAlarmHistory.AlarmHistoryItems)
```

```
        {
            alarmHistory.Add(data);
        }
        return alarmHistory;
    }

    /// <summary>
    /// Delete a list of alarms from CloudWatch.
    /// </summary>
    /// <param name="alarmNames">A list of names of alarms to delete.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteAlarms(List<string> alarmNames)
    {
        var deleteAlarmsResult = await _amazonCloudWatch.DeleteAlarmsAsync(
            new DeleteAlarmsRequest()
            {
                AlarmNames = alarmNames
            });

        return deleteAlarmsResult.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Disable the actions for a list of alarms from CloudWatch.
    /// </summary>
    /// <param name="alarmNames">A list of names of alarms.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DisableAlarmActions(List<string> alarmNames)
    {
        var disableAlarmActionsResult = await
 _amazonCloudWatch.DisableAlarmActionsAsync(
            new DisableAlarmActionsRequest()
            {
                AlarmNames = alarmNames
            });

        return disableAlarmActionsResult.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Enable the actions for a list of alarms from CloudWatch.
    /// </summary>
    /// <param name="alarmNames">A list of names of alarms.</param>
    /// <returns>True if successful.</returns>
```

```
    public async Task<bool> EnableAlarmActions(List<string> alarmNames)
    {
        var enableAlarmActionsResult = await
 _amazonCloudWatch.EnableAlarmActionsAsync(
            new EnableAlarmActionsRequest()
            {
                AlarmNames = alarmNames
            });

        return enableAlarmActionsResult.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Add an anomaly detector for a single metric.
    /// </summary>
    /// <param name="anomalyDetector">A single metric anomaly detector.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> PutAnomalyDetector(SingleMetricAnomalyDetector
 anomalyDetector)
    {
        var putAlarmDetectorResult = await
 _amazonCloudWatch.PutAnomalyDetectorAsync(
            new PutAnomalyDetectorRequest()
            {
                SingleMetricAnomalyDetector = anomalyDetector
            });

        return putAlarmDetectorResult.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Describe anomaly detectors for a metric and namespace.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metricName">The metric of the anomaly detectors.</param>
    /// <returns>The list of detectors.</returns>
    public async Task<List<AnomalyDetector>> DescribeAnomalyDetectors(string
 metricNamespace, string metricName)
    {
        List<AnomalyDetector> detectors = new List<AnomalyDetector>();
        var paginatedDescribeAnomalyDetectors =
 _amazonCloudWatch.Paginators.DescribeAnomalyDetectors(
            new DescribeAnomalyDetectorsRequest()
            {
```

```
                MetricName = metricName,
                Namespace = metricNamespace
            });

        await foreach (var data in
 paginatedDescribeAnomalyDetectors.AnomalyDetectors)
        {
            detectors.Add(data);
        }

        return detectors;
    }

    /// <summary>
    /// Delete a single metric anomaly detector.
    /// </summary>
    /// <param name="anomalyDetector">The anomaly detector to delete.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteAnomalyDetector(SingleMetricAnomalyDetector
 anomalyDetector)
    {
        var deleteAnomalyDetectorResponse = await
 _amazonCloudWatch.DeleteAnomalyDetectorAsync(
            new DeleteAnomalyDetectorRequest()
            {
                SingleMetricAnomalyDetector = anomalyDetector
            });

        return deleteAnomalyDetectorResponse.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Delete a list of CloudWatch dashboards.
    /// </summary>
    /// <param name="dashboardNames">List of dashboard names to delete.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteDashboards(List<string> dashboardNames)
    {
        var deleteDashboardsResponse = await
 _amazonCloudWatch.DeleteDashboardsAsync(
            new DeleteDashboardsRequest()
            {
                DashboardNames = dashboardNames
            });
```

```
        return deleteDashboardsResponse.HttpStatusCode == HttpStatusCode.OK;
    }
}
```

- For API details, see the following topics in *Amazon SDK for .NET API Reference*.
  - DeleteAlarms
  - DeleteAnomalyDetector
  - DeleteDashboards
  - DescribeAlarmHistory
  - DescribeAlarms
  - DescribeAlarmsForMetric
  - DescribeAnomalyDetectors
  - GetMetricData
  - GetMetricStatistics
  - GetMetricWidgetImage
  - ListMetrics
  - PutAnomalyDetector
  - PutDashboard
  - PutMetricAlarm
  - PutMetricData

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

Run an interactive scenario demonstrating CloudWatch features.

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import software.amazon.awssdk.services.cloudwatch.model.CloudWatchException;
import
 software.amazon.awssdk.services.cloudwatch.model.DashboardInvalidInputErrorException;
import software.amazon.awssdk.services.cloudwatch.model.DeleteAlarmsResponse;
import
 software.amazon.awssdk.services.cloudwatch.model.DeleteAnomalyDetectorResponse;
import software.amazon.awssdk.services.cloudwatch.model.DeleteDashboardsResponse;
import software.amazon.awssdk.services.cloudwatch.model.Dimension;
import
 software.amazon.awssdk.services.cloudwatch.model.GetMetricStatisticsResponse;
import software.amazon.awssdk.services.cloudwatch.model.LimitExceededException;
import software.amazon.awssdk.services.cloudwatch.model.PutDashboardResponse;
import software.amazon.awssdk.services.cloudwatch.model.PutMetricDataResponse;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Scanner;
import java.util.concurrent.CompletableFuture;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * To enable billing metrics and statistics for this example, make sure billing
 * alerts are enabled for your account:
 * https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/
monitor_estimated_charges_with_cloudwatch.html#turning_on_billing_metrics
 *
 * This Java code example performs the following tasks:
 *
 * 1. List available namespaces from Amazon CloudWatch.
 * 2. List available metrics within the selected Namespace.
 * 3. Get statistics for the selected metric over the last day.
 * 4. Get CloudWatch estimated billing for the last week.
 * 5. Create a new CloudWatch dashboard with metrics.
 * 6. List dashboards using a paginator.
 * 7. Create a new custom metric by adding data for it.
```

```
 * 8. Add the custom metric to the dashboard.
 * 9. Create an alarm for the custom metric.
 * 10. Describe current alarms.
 * 11. Get current data for the new custom metric.
 * 12. Push data into the custom metric to trigger the alarm.
 * 13. Check the alarm state using the action DescribeAlarmsForMetric.
 * 14. Get alarm history for the new alarm.
 * 15. Add an anomaly detector for the custom metric.
 * 16. Describe current anomaly detectors.
 * 17. Get a metric image for the custom metric.
 * 18. Clean up the Amazon CloudWatch resources.
 */
public class CloudWatchScenario {
    public static final String DASHES = new String(new char[80]).replace("\0",
 "-");

    static CloudWatchActions cwActions = new CloudWatchActions();

    private static final Logger logger =
 LoggerFactory.getLogger(CloudWatchScenario.class);
    static Scanner scanner = new Scanner(System.in);
    public static void main(String[] args) throws Throwable {

        final String usage = """

            Usage:
              <myDate> <costDateWeek> <dashboardName> <dashboardJson>
 <dashboardAdd> <settings> <metricImage> \s

            Where:
              myDate - The start date to use to get metric statistics. (For
 example, 2023-01-11T18:35:24.00Z.)\s
              costDateWeek - The start date to use to get AWS/Billing statistics.
 (For example, 2023-01-11T18:35:24.00Z.)\s
              dashboardName - The name of the dashboard to create.\s
              dashboardJson - The location of a JSON file to use to create a
 dashboard. (See jsonWidgets.json in javav2/example_code/cloudwatch.)\s
              dashboardAdd - The location of a JSON file to use to update a
 dashboard. (See CloudDashboard.json in javav2/example_code/cloudwatch.)\s
              settings - The location of a JSON file from which various values
 are read. (See settings.json in javav2/example_code/cloudwatch.)\s
              metricImage - The location of a BMP file that is used to create a
 graph.\s
            """;
```

```
        if (args.length != 7) {
            logger.info(usage);
            return;
        }
        String myDate = args[0];
        String costDateWeek = args[1];
        String dashboardName = args[2];
        String dashboardJson = args[3];
        String dashboardAdd = args[4];
        String settings = args[5];
        String metricImage = args[6];

        logger.info(DASHES);
        logger.info("Welcome to the Amazon CloudWatch Basics scenario.");
        logger.info("""
            Amazon CloudWatch is a comprehensive monitoring and observability
 service
            provided by Amazon Web Services (AWS). It is designed to help you
 monitor your
            AWS resources, applications, and services, as well as on-premises
 resources,
            in real-time.

            CloudWatch collects and tracks various types of data, including
 metrics,
            logs, and events, from your AWS and on-premises resources. It allows
 you to set
            alarms and automatically respond to changes in your environment,
            enabling you to quickly identify and address issues before they
 impact your
            applications or services.

            With CloudWatch, you can gain visibility into your entire
 infrastructure, from the cloud
            to the edge, and use this information to make informed decisions and
 optimize your
            resource utilization.

            This scenario guides you through how to perform Amazon CloudWatch
 tasks by using the
            AWS SDK for Java v2. Let's get started...
            """);
        waitForInputToContinue(scanner);
```

```
        try {
            runScenario(myDate, costDateWeek, dashboardName, dashboardJson,
dashboardAdd, settings, metricImage);
        } catch (RuntimeException e) {
            e.printStackTrace();
        }
        logger.info(DASHES);
    }

    private static void runScenario(String myDate, String costDateWeek, String
dashboardName, String dashboardJson, String dashboardAdd, String settings,
String metricImage ) throws Throwable {
        Double dataPoint = Double.parseDouble("10.0");
        logger.info(DASHES);
        logger.info("""
        1. List at least five available unique namespaces from Amazon
CloudWatch.
        Select one from the list.
        """);
        String selectedNamespace;
        String selectedMetrics;
        int num;
        try {
            CompletableFuture<ArrayList<String>> future =
cwActions.listNameSpacesAsync();
            ArrayList<String> list = future.join();
            for (int z = 0; z < 5; z++) {
                int index = z + 1;
                logger.info("    " + index + ". {}", list.get(z));
            }

            num = Integer.parseInt(scanner.nextLine());
            if (1 <= num && num <= 5) {
                selectedNamespace = list.get(num - 1);
            } else {
                logger.info("You did not select a valid option.");
                return;
            }
            logger.info("You selected {}", selectedNamespace);

        } catch (RuntimeException rt) {
            Throwable cause = rt.getCause();
            if (cause instanceof CloudWatchException cwEx) {
```

```
                logger.info("CloudWatch error occurred: Error message: {}, Error
code {}", cwEx.getMessage(), cwEx.awsErrorDetails().errorCode());
            } else {
                logger.info("An unexpected error occurred: " + rt.getMessage());
            }
            throw cause;
        }
        waitForInputToContinue(scanner);
        logger.info(DASHES);

        logger.info(DASHES);
        logger.info("2. List available metrics within the selected namespace.");
        logger.info("""
            A metric is a measure of the performance or health of your AWS
resources,
            applications, or custom resources. Metrics are the basic building
blocks of CloudWatch
            and provide data points that represent a specific aspect of your
system or application over time.

            Select a metric from the list.
            """);

        Dimension myDimension = null;
        try {
            CompletableFuture<ArrayList<String>> future =
cwActions.listMetsAsync(selectedNamespace);
            ArrayList<String> metList = future.join();
            logger.info("Metrics successfully retrieved. Total metrics: {}",
metList.size());
            for (int z = 0; z < 5; z++) {
                int index = z + 1;
                logger.info("    " + index + ". " + metList.get(z));
            }
            num = Integer.parseInt(scanner.nextLine());
            if (1 <= num && num <= 5) {
                selectedMetrics = metList.get(num - 1);
            } else {
                logger.info("You did not select a valid option.");
                return;
            }
            logger.info("You selected {}", selectedMetrics);

        } catch (RuntimeException rt) {
```

```
            Throwable cause = rt.getCause();
            if (cause instanceof CloudWatchException cwEx) {
                logger.info("CloudWatch error occurred: Error message: {}, Error
code {}", cwEx.getMessage(), cwEx.awsErrorDetails().errorCode());
            } else {
                logger.info("An unexpected error occurred: {}", rt.getMessage());
            }
            throw cause;
        }

        try {
            myDimension =
cwActions.getSpecificMetAsync(selectedNamespace).join();
            logger.info("Metric statistics successfully retrieved and
displayed.");
        } catch (RuntimeException rt) {
            Throwable cause = rt.getCause();
            if (cause instanceof CloudWatchException cwEx) {
                logger.info("CloudWatch error occurred: Error message: {}, Error
code {}", cwEx.getMessage(), cwEx.awsErrorDetails().errorCode());
            } else {
                logger.info("An unexpected error occurred: {}", rt.getMessage());
            }
            throw cause;
        }

        waitForInputToContinue(scanner);
        logger.info(DASHES);

        logger.info(DASHES);
        logger.info("3. Get statistics for the selected metric over the last
day.");
        logger.info("""
            Statistics refer to the various mathematical calculations that can be
performed on the
            collected metrics to derive meaningful insights. Statistics provide a
way to summarize and
            analyze the data collected for a specific metric over a specified
time period.
            """);
        waitForInputToContinue(scanner);
        String metricOption = "";
        ArrayList<String> statTypes = new ArrayList<>();
        statTypes.add("SampleCount");
```

```
        statTypes.add("Average");
        statTypes.add("Sum");
        statTypes.add("Minimum");
        statTypes.add("Maximum");

        for (int t = 0; t < 5; t++) {
            logger.info("    " + (t + 1) + ". {}", statTypes.get(t));
        }
        logger.info("Select a metric statistic by entering a number from the
 preceding list:");
        num = Integer.parseInt(scanner.nextLine());
        if (1 <= num && num <= 5) {
            metricOption = statTypes.get(num - 1);
        } else {
            logger.info("You did not select a valid option.");
            return;
        }
        logger.info("You selected " + metricOption);
        waitForInputToContinue(scanner);
        try {
            CompletableFuture<GetMetricStatisticsResponse> future =
 cwActions.getAndDisplayMetricStatisticsAsync(selectedNamespace, selectedMetrics,
 metricOption, myDate, myDimension);
            future.join();
            logger.info("Metric statistics retrieved successfully.");

        } catch (RuntimeException rt) {
            Throwable cause = rt.getCause();
            if (cause instanceof CloudWatchException cwEx) {
                logger.info("CloudWatch error occurred: Error message: {}, Error
 code {}", cwEx.getMessage(), cwEx.awsErrorDetails().errorCode());
            } else {
                logger.info("An unexpected error occurred: {}", rt.getMessage());
            }
            throw cause;
        }
        waitForInputToContinue(scanner);
        logger.info(DASHES);

        logger.info(DASHES);
        logger.info("4. Get CloudWatch estimated billing for the last week.");
        waitForInputToContinue(scanner);
         try {
```

```
            CompletableFuture<GetMetricStatisticsResponse> future =
 cwActions.getMetricStatisticsAsync(costDateWeek);
            future.join();

            logger.info("Metric statistics successfully retrieved and
 displayed.");
        } catch (RuntimeException rt) {
            Throwable cause = rt.getCause();
            if (cause instanceof CloudWatchException cwEx) {
                logger.info("CloudWatch error occurred: Error message: {}, Error
 code {}", cwEx.getMessage(), cwEx.awsErrorDetails().errorCode());
            } else {
                logger.info("An unexpected error occurred: {}", rt.getMessage());
            }
             throw cause;
         }
        waitForInputToContinue(scanner);
        logger.info(DASHES);

        logger.info(DASHES);
        logger.info("5. Create a new CloudWatch dashboard with metrics.");
        waitForInputToContinue(scanner);
        try {
            CompletableFuture<PutDashboardResponse> future =
 cwActions.createDashboardWithMetricsAsync(dashboardName, dashboardJson);
            future.join();

        } catch (RuntimeException | IOException rt) {
            Throwable cause = rt.getCause();
            if (cause instanceof DashboardInvalidInputErrorException cwEx) {
                logger.info("Invalid CloudWatch data. Error message: {}, Error
 code {}", cwEx.getMessage(), cwEx.awsErrorDetails().errorCode());
            } else {
                logger.info("An unexpected error occurred: {}", rt.getMessage());
            }
            throw cause;
        }
        waitForInputToContinue(scanner);
        logger.info(DASHES);

        logger.info(DASHES);
        logger.info("6. List dashboards using a paginator.");
        waitForInputToContinue(scanner);
        try {
```

```
            CompletableFuture<Void> future = cwActions.listDashboardsAsync();
            future.join();

        } catch (RuntimeException rt) {
            Throwable cause = rt.getCause();
            if (cause instanceof CloudWatchException cwEx) {
                logger.info("CloudWatch error occurred: Error message: {}, Error
 code {}", cwEx.getMessage(), cwEx.awsErrorDetails().errorCode());
            } else {
                logger.info("An unexpected error occurred: {}", rt.getMessage());
            }
            throw cause;
        }
        waitForInputToContinue(scanner);
        logger.info(DASHES);

        logger.info(DASHES);
        logger.info("7. Create a new custom metric by adding data to it.");
        logger.info("""
            The primary benefit of using a custom metric in Amazon CloudWatch is
 the ability to
            monitor and collect data that is specific to your application or
 infrastructure.
            """);
        waitForInputToContinue(scanner);
        try {
            CompletableFuture<PutMetricDataResponse> future =
 cwActions.createNewCustomMetricAsync(dataPoint);
            future.join();

        } catch (RuntimeException rt) {
            Throwable cause = rt.getCause();
            if (cause instanceof CloudWatchException cwEx) {
                logger.info("CloudWatch error occurred: Error message: {}, Error
 code {}", cwEx.getMessage(), cwEx.awsErrorDetails().errorCode());
            } else {
                logger.info("An unexpected error occurred: {}", rt.getMessage());
            }
            throw cause;
        }
        waitForInputToContinue(scanner);
        logger.info(DASHES);

        logger.info(DASHES);
```

```
        logger.info("8. Add an additional metric to the dashboard.");
        waitForInputToContinue(scanner);
        try {
            CompletableFuture<PutDashboardResponse> future =
 cwActions.addMetricToDashboardAsync(dashboardAdd, dashboardName);
            future.join();

        } catch (RuntimeException rt) {
            Throwable cause = rt.getCause();
            if (cause instanceof DashboardInvalidInputErrorException cwEx) {
                logger.info("Invalid CloudWatch data. Error message: {}, Error
 code {}", cwEx.getMessage(), cwEx.awsErrorDetails().errorCode());
            } else {
                logger.info("An unexpected error occurred: {}", rt.getMessage());
            }
            throw cause;
        }
        logger.info(DASHES);

        logger.info(DASHES);
        logger.info("9. Create an alarm for the custom metric.");
        waitForInputToContinue(scanner);
        String alarmName = "" ;
        try {
            CompletableFuture<String> future =
 cwActions.createAlarmAsync(settings);
            alarmName = future.join();

        } catch (RuntimeException rt) {
            Throwable cause = rt.getCause();
            if (cause instanceof LimitExceededException cwEx) {
                logger.info("The quota for alarms has been
 reached: Error message: {}, Error code {}", cwEx.getMessage(),
 cwEx.awsErrorDetails().errorCode());
            } else {
                logger.info("An unexpected error occurred: {}", rt.getMessage());
            }
            throw cause;
        }
        waitForInputToContinue(scanner);
        logger.info(DASHES);

        logger.info(DASHES);
        logger.info("10. Describe ten current alarms.");
```

```
        waitForInputToContinue(scanner);
        try {
            CompletableFuture<Void> future = cwActions.describeAlarmsAsync();
            future.join();

        } catch (RuntimeException rt) {
            Throwable cause = rt.getCause();
            if (cause instanceof CloudWatchException cwEx) {
                logger.info("CloudWatch error occurred: Error message: {}, Error
 code {}", cwEx.getMessage(), cwEx.awsErrorDetails().errorCode());
            } else {
                logger.info("An unexpected error occurred: {}", rt.getMessage());
            }
            throw cause;
        }
        waitForInputToContinue(scanner);
        logger.info(DASHES);

        logger.info(DASHES);
        logger.info("11. Get current data for new custom metric.");
        try {
            CompletableFuture<Void> future =
 cwActions.getCustomMetricDataAsync(settings);
            future.join();

        } catch (RuntimeException rt) {
            Throwable cause = rt.getCause();
            if (cause instanceof CloudWatchException cwEx) {
                logger.info("CloudWatch error occurred: Error message: {}, Error
 code {}", cwEx.getMessage(), cwEx.awsErrorDetails().errorCode());
            } else {
                logger.info("An unexpected error occurred: {}", rt.getMessage());
            }
            throw cause;
        }
        waitForInputToContinue(scanner);
        logger.info(DASHES);

        logger.info(DASHES);
        logger.info("12. Push data into the custom metric to trigger the
 alarm.");
        waitForInputToContinue(scanner);
        try {
```

```
            CompletableFuture<PutMetricDataResponse> future =
cwActions.addMetricDataForAlarmAsync(settings);
            future.join();

        } catch (RuntimeException rt) {
            Throwable cause = rt.getCause();
            if (cause instanceof CloudWatchException cwEx) {
                logger.info("CloudWatch error occurred: Error message: {}, Error
code {}", cwEx.getMessage(), cwEx.awsErrorDetails().errorCode());
            } else {
                logger.info("An unexpected error occurred: {}", rt.getMessage());
            }
            throw cause;
        }
        waitForInputToContinue(scanner);
        logger.info(DASHES);

        logger.info(DASHES);
        logger.info("13. Check the alarm state using the action
DescribeAlarmsForMetric.");
        waitForInputToContinue(scanner);
        try {
            CompletableFuture<Void> future =
cwActions.checkForMetricAlarmAsync(settings);
            future.join();

        } catch (RuntimeException rt) {
            Throwable cause = rt.getCause();
            if (cause instanceof CloudWatchException cwEx) {
                logger.info("CloudWatch error occurred: Error message: {}, Error
code {}", cwEx.getMessage(), cwEx.awsErrorDetails().errorCode());
            } else {
                logger.info("An unexpected error occurred: {}", rt.getMessage());
            }
            throw cause;
        }
        waitForInputToContinue(scanner);
        logger.info(DASHES);

        logger.info(DASHES);
        logger.info("14. Get alarm history for the new alarm.");
        waitForInputToContinue(scanner);
        try {
```

```
            CompletableFuture<Void> future =
cwActions.getAlarmHistoryAsync(settings, myDate);
            future.join();

        } catch (RuntimeException rt) {
            Throwable cause = rt.getCause();
            if (cause instanceof CloudWatchException cwEx) {
                logger.info("CloudWatch error occurred: Error message: {}, Error
code {}", cwEx.getMessage(), cwEx.awsErrorDetails().errorCode());
            } else {
                logger.info("An unexpected error occurred: {}", rt.getMessage());
            }
            throw cause;
        }
        logger.info(DASHES);

        logger.info(DASHES);
        logger.info("15. Add an anomaly detector for the custom metric.");
        logger.info("""
            An anomaly detector is a feature that automatically detects unusual
patterns or deviations in your
            monitored metrics. It uses machine learning algorithms to analyze the
historical behavior
            of your metrics and establish a baseline.

            The anomaly detector then compares the current metric values against
this baseline and
            identifies any anomalies or outliers that may indicate potential
issues or unexpected changes
            in your system's performance or behavior.

            """);
        waitForInputToContinue(scanner);
        try {
            CompletableFuture<Void> future =
cwActions.addAnomalyDetectorAsync(settings);
            future.join();

        } catch (RuntimeException rt) {
            Throwable cause = rt.getCause();
            if (cause instanceof CloudWatchException cwEx) {
                logger.info("CloudWatch error occurred: Error message: {}, Error
code {}", cwEx.getMessage(), cwEx.awsErrorDetails().errorCode());
            } else {
```

```
                logger.info("An unexpected error occurred: {}", rt.getMessage());
            }
            throw cause;
        }
        waitForInputToContinue(scanner);
        logger.info(DASHES);

        logger.info(DASHES);
        logger.info("16. Describe current anomaly detectors.");
        waitForInputToContinue(scanner);
        try {
            CompletableFuture<Void> future =
 cwActions.describeAnomalyDetectorsAsync(settings);
            future.join();

        } catch (RuntimeException rt) {
            Throwable cause = rt.getCause();
            if (cause instanceof CloudWatchException cwEx) {
                logger.info("CloudWatch error occurred: Error message: {}, Error
 code {}", cwEx.getMessage(), cwEx.awsErrorDetails().errorCode());
            } else {
                logger.info("An unexpected error occurred: {}", rt.getMessage());
            }
            throw cause;
        }
        waitForInputToContinue(scanner);
        logger.info(DASHES);

        logger.info(DASHES);
        logger.info("17. Get a metric image for the custom metric.");
        try {
            CompletableFuture<Void> future =
 cwActions.downloadAndSaveMetricImageAsync(metricImage);
            future.join();

        } catch (RuntimeException rt) {
            Throwable cause = rt.getCause();
            if (cause instanceof CloudWatchException cwEx) {
                logger.info("CloudWatch error occurred: Error message: {}, Error
 code {}", cwEx.getMessage(), cwEx.awsErrorDetails().errorCode());
            } else {
                logger.info("An unexpected error occurred: {}", rt.getMessage());
            }
            throw cause;
```

```
        }
        logger.info(DASHES);

        logger.info(DASHES);
        logger.info("18. Clean up the Amazon CloudWatch resources.");

        try {
            logger.info(". Delete the Dashboard.");
            waitForInputToContinue(scanner);
            CompletableFuture<DeleteDashboardsResponse> future =
cwActions.deleteDashboardAsync(dashboardName);
            future.join();

        } catch (RuntimeException rt) {
            Throwable cause = rt.getCause();
            if (cause instanceof CloudWatchException cwEx) {
                logger.info("CloudWatch error occurred: Error message: {}, Error
code {}", cwEx.getMessage(), cwEx.awsErrorDetails().errorCode());
            } else {
                logger.info("An unexpected error occurred: {}", rt.getMessage());
            }
            throw cause;
        }

        try {
            logger.info("Delete the alarm.");
            waitForInputToContinue(scanner);
            CompletableFuture<DeleteAlarmsResponse> future =
cwActions.deleteCWAlarmAsync(alarmName);
            future.join();

        } catch (RuntimeException rt) {
            Throwable cause = rt.getCause();
            if (cause instanceof CloudWatchException cwEx) {
                logger.info("CloudWatch error occurred: Error message: {}, Error
code {}", cwEx.getMessage(), cwEx.awsErrorDetails().errorCode());
            } else {
                logger.info("An unexpected error occurred: {}", rt.getMessage());
            }
            throw cause;
        }

        try {
            logger.info("Delete the anomaly detector.");
```

```
            waitForInputToContinue(scanner);
            CompletableFuture<DeleteAnomalyDetectorResponse> future =
 cwActions.deleteAnomalyDetectorAsync(settings);
            future.join();

        } catch (RuntimeException rt) {
            Throwable cause = rt.getCause();
            if (cause instanceof CloudWatchException cwEx) {
                logger.info("CloudWatch error occurred: Error message: {}, Error
 code {}", cwEx.getMessage(), cwEx.awsErrorDetails().errorCode());
            } else {
                logger.info("An unexpected error occurred: {}", rt.getMessage());
            }
            throw cause;
        }
        waitForInputToContinue(scanner);
        logger.info(DASHES);

        logger.info(DASHES);
        logger.info("The Amazon CloudWatch example scenario is complete.");
        logger.info(DASHES);
    }

    private static void waitForInputToContinue(Scanner scanner) {
        while (true) {
            logger.info("");
            logger.info("Enter 'c' followed by <ENTER> to continue:");
            String input = scanner.nextLine();
            if (input.trim().equalsIgnoreCase("c")) {
                logger.info("Continuing with the program...");
                logger.info("");
                break;
            } else {
                // Handle invalid input.
                logger.info("Invalid input. Please try again.");
            }
        }
    }
}
```

A wrapper class for CloudWatch SDK methods.

```
public class CloudWatchActions {

    private static CloudWatchAsyncClient cloudWatchAsyncClient;

    private static final Logger logger =
 LoggerFactory.getLogger(CloudWatchActions.class);

    /**
     * Retrieves an asynchronous CloudWatch client instance.
     *
     * <p>
     * This method ensures that the CloudWatch client is initialized with the
 following configurations:
     * <ul>
     *     <li>Maximum concurrency: 100</li>
     *     <li>Connection timeout: 60 seconds</li>
     *     <li>Read timeout: 60 seconds</li>
     *     <li>Write timeout: 60 seconds</li>
     *     <li>API call timeout: 2 minutes</li>
     *     <li>API call attempt timeout: 90 seconds</li>
     *     <li>Retry strategy: STANDARD</li>
     * </ul>
     * </p>
     *
     * @return the asynchronous CloudWatch client instance
     */
    private static CloudWatchAsyncClient getAsyncClient() {
        if (cloudWatchAsyncClient == null) {
            SdkAsyncHttpClient httpClient = NettyNioAsyncHttpClient.builder()
                .maxConcurrency(100)
                .connectionTimeout(Duration.ofSeconds(60))
                .readTimeout(Duration.ofSeconds(60))
                .writeTimeout(Duration.ofSeconds(60))
                .build();

            ClientOverrideConfiguration overrideConfig =
 ClientOverrideConfiguration.builder()
                .apiCallTimeout(Duration.ofMinutes(2))
                .apiCallAttemptTimeout(Duration.ofSeconds(90))
                .retryStrategy(RetryMode.STANDARD)
                .build();

            cloudWatchAsyncClient = CloudWatchAsyncClient.builder()
```

```
                .httpClient(httpClient)
                .overrideConfiguration(overrideConfig)
                .build();
        }
        return cloudWatchAsyncClient;
    }

    /**
     * Deletes an Anomaly Detector.
     *
     * @param fileName the name of the file containing the Anomaly Detector 
configuration
     * @return a CompletableFuture that represents the asynchronous deletion of 
the Anomaly Detector
     */
    public CompletableFuture<DeleteAnomalyDetectorResponse> 
deleteAnomalyDetectorAsync(String fileName) {
        CompletableFuture<JsonNode> readFileFuture = 
CompletableFuture.supplyAsync(() -> {
            try {
                JsonParser parser = new JsonFactory().createParser(new 
File(fileName));
                return new ObjectMapper().readTree(parser); // Return the root 
node
            } catch (IOException e) {
                throw new RuntimeException("Failed to read or parse the file", 
e);
            }
        });

        return readFileFuture.thenCompose(rootNode -> {
            String customMetricNamespace = 
rootNode.findValue("customMetricNamespace").asText();
            String customMetricName = 
rootNode.findValue("customMetricName").asText();

            SingleMetricAnomalyDetector singleMetricAnomalyDetector = 
SingleMetricAnomalyDetector.builder()
                .metricName(customMetricName)
                .namespace(customMetricNamespace)
                .stat("Maximum")
                .build();
```

```
            DeleteAnomalyDetectorRequest request =
DeleteAnomalyDetectorRequest.builder()
                .singleMetricAnomalyDetector(singleMetricAnomalyDetector)
                .build();

            return getAsyncClient().deleteAnomalyDetector(request);
        }).whenComplete((result, exception) -> {
            if (exception != null) {
                throw new RuntimeException("Failed to delete the Anomaly
Detector", exception);
            } else {
                logger.info("Successfully deleted the Anomaly Detector.");
            }
        });
    }

    /**
     * Deletes a CloudWatch alarm.
     *
     * @param alarmName the name of the alarm to be deleted
     * @return a {@link CompletableFuture} representing the asynchronous
operation to delete the alarm
     * the {@link DeleteAlarmsResponse} is returned when the operation completes
successfully,
     * or a {@link RuntimeException} is thrown if the operation fails
     */
    public CompletableFuture<DeleteAlarmsResponse> deleteCWAlarmAsync(String
alarmName) {
        DeleteAlarmsRequest request = DeleteAlarmsRequest.builder()
            .alarmNames(alarmName)
            .build();

        return getAsyncClient().deleteAlarms(request)
            .whenComplete((response, exception) -> {
                if (exception != null) {
                    throw new RuntimeException("Failed to delete the alarm:{} " +
alarmName, exception);
                } else {
                    logger.info("Successfully deleted alarm {} ", alarmName);
                }
            });
    }

    /**
```

```
     * Deletes the specified dashboard.
     *
     * @param dashboardName the name of the dashboard to be deleted
     * @return a {@link CompletableFuture} representing the asynchronous
 operation of deleting the dashboard
     * @throws RuntimeException if the dashboard deletion fails
     */
    public CompletableFuture<DeleteDashboardsResponse>
 deleteDashboardAsync(String dashboardName) {
        DeleteDashboardsRequest dashboardsRequest =
 DeleteDashboardsRequest.builder()
            .dashboardNames(dashboardName)
            .build();

        return getAsyncClient().deleteDashboards(dashboardsRequest)
            .whenComplete((response, exception) -> {
                if (exception != null) {
                    throw new RuntimeException("Failed to delete the dashboard: "
 + dashboardName, exception);
                } else {
                    logger.info("{} was successfully deleted.", dashboardName);
                }
            });
    }


    /**
     * Retrieves and saves a custom metric image to a file.
     *
     * @param fileName the name of the file to save the metric image to
     * @return a {@link CompletableFuture} that completes when the image has been
 saved to the file
     */
    public CompletableFuture<Void> downloadAndSaveMetricImageAsync(String
 fileName) {
        logger.info("Getting Image data for custom metric.");
        String myJSON = """
              {
                  "title": "Example Metric Graph",
                  "view": "timeSeries",
                  "stacked ": false,
                  "period": 10,
                  "width": 1400,
                  "height": 600,
```

```
                   "metrics": [
                       [
                       "AWS/Billing",
                       "EstimatedCharges",
                       "Currency",
                       "USD"
                      ]
                   ]
               }
             """;

        GetMetricWidgetImageRequest imageRequest =
GetMetricWidgetImageRequest.builder()
            .metricWidget(myJSON)
            .build();

        return getAsyncClient().getMetricWidgetImage(imageRequest)
            .thenCompose(response -> {
                SdkBytes sdkBytes = response.metricWidgetImage();
                byte[] bytes = sdkBytes.asByteArray();
                return CompletableFuture.runAsync(() -> {
                    try {
                        File outputFile = new File(fileName);
                        try (FileOutputStream outputStream = new
FileOutputStream(outputFile)) {
                            outputStream.write(bytes);
                        }
                    } catch (IOException e) {
                        throw new RuntimeException("Failed to write image to
file", e);
                    }
                });
            })
            .whenComplete((result, exception) -> {
                if (exception != null) {
                    throw new RuntimeException("Error getting and saving metric
image", exception);
                } else {
                    logger.info("Image data saved successfully to {}", fileName);
                }
            });
    }
```

```
   /**
     * Describes the anomaly detectors based on the specified JSON file.
     *
     * @param fileName the name of the JSON file containing the custom metric
 namespace and name
     * @return a {@link CompletableFuture} that completes when the anomaly
 detectors have been described
     * @throws RuntimeException if there is a failure during the operation, such
 as when reading or parsing the JSON file,
     *                          or when describing the anomaly detectors
     */
   public CompletableFuture<Void> describeAnomalyDetectorsAsync(String fileName)
 {
       CompletableFuture<JsonNode> readFileFuture =
 CompletableFuture.supplyAsync(() -> {
           try {
               JsonParser parser = new JsonFactory().createParser(new
 File(fileName));
               return new ObjectMapper().readTree(parser);
           } catch (IOException e) {
               throw new RuntimeException("Failed to read or parse the file",
 e);
           }
       });

       return readFileFuture.thenCompose(rootNode -> {
           try {
               String customMetricNamespace =
 rootNode.findValue("customMetricNamespace").asText();
               String customMetricName =
 rootNode.findValue("customMetricName").asText();

               DescribeAnomalyDetectorsRequest detectorsRequest =
 DescribeAnomalyDetectorsRequest.builder()
                   .maxResults(10)
                   .metricName(customMetricName)
                   .namespace(customMetricNamespace)
                   .build();

               return
 getAsyncClient().describeAnomalyDetectors(detectorsRequest).thenAccept(response
 -> {
                   List<AnomalyDetector> anomalyDetectorList =
 response.anomalyDetectors();
```

```
                   for (AnomalyDetector detector : anomalyDetectorList) {
                       logger.info("Metric name: {} ",
detector.singleMetricAnomalyDetector().metricName());
                       logger.info("State: {} ", detector.stateValue());
                   }
               });
           } catch (RuntimeException e) {
               throw new RuntimeException("Failed to describe anomaly
detectors", e);
           }
       }).whenComplete((result, exception) -> {
           if (exception != null) {
               throw new RuntimeException("Error describing anomaly detectors",
exception);
           }
       });
   }


   /**
    * Adds an anomaly detector for the given file.
    *
    * @param fileName the name of the file containing the anomaly detector
configuration
    * @return a {@link CompletableFuture} that completes when the anomaly
detector has been added
    */
   public CompletableFuture<Void> addAnomalyDetectorAsync(String fileName) {
       CompletableFuture<JsonNode> readFileFuture =
CompletableFuture.supplyAsync(() -> {
           try {
               JsonParser parser = new JsonFactory().createParser(new
File(fileName));
               return new ObjectMapper().readTree(parser); // Return the root
node
           } catch (IOException e) {
               throw new RuntimeException("Failed to read or parse the file",
e);
           }
       });

       return readFileFuture.thenCompose(rootNode -> {
           try {
```

```
                String customMetricNamespace =
rootNode.findValue("customMetricNamespace").asText();
                String customMetricName =
rootNode.findValue("customMetricName").asText();

                SingleMetricAnomalyDetector singleMetricAnomalyDetector =
SingleMetricAnomalyDetector.builder()
                    .metricName(customMetricName)
                    .namespace(customMetricNamespace)
                    .stat("Maximum")
                    .build();

                PutAnomalyDetectorRequest anomalyDetectorRequest =
PutAnomalyDetectorRequest.builder()
                    .singleMetricAnomalyDetector(singleMetricAnomalyDetector)
                    .build();

                return
getAsyncClient().putAnomalyDetector(anomalyDetectorRequest).thenAccept(response
-> {
                    logger.info("Added anomaly detector for metric {}",
customMetricName);
                });
            } catch (Exception e) {
                throw new RuntimeException("Failed to create anomaly detector",
e);
            }
        }).whenComplete((result, exception) -> {
            if (exception != null) {
                throw new RuntimeException("Error adding anomaly detector",
exception);
            }
        });
    }


    /**
     * Retrieves the alarm history for a given alarm name and date range.
     *
     * @param fileName the path to the JSON file containing the alarm name
     * @param date     the date to start the alarm history search (in the format
"yyyy-MM-dd'T'HH:mm:ss'Z'")
     * @return a {@code CompletableFuture<Void>} that completes when the alarm
history has been retrieved and processed
```

```
     */
    public CompletableFuture<Void> getAlarmHistoryAsync(String fileName, String 
date) {
        CompletableFuture<String> readFileFuture = 
CompletableFuture.supplyAsync(() -> {
            try {
                JsonParser parser = new JsonFactory().createParser(new 
File(fileName));
                com.fasterxml.jackson.databind.JsonNode rootNode = new 
ObjectMapper().readTree(parser);
                return rootNode.findValue("exampleAlarmName").asText(); // Return 
alarmName from the JSON file
            } catch (IOException e) {
                throw new RuntimeException("Failed to read or parse the file", 
e);
            }
        });

        // Use the alarm name to describe alarm history with a paginator.
        return readFileFuture.thenCompose(alarmName -> {
            try {
                Instant start = Instant.parse(date);
                Instant endDate = Instant.now();
                DescribeAlarmHistoryRequest historyRequest = 
DescribeAlarmHistoryRequest.builder()
                    .startDate(start)
                    .endDate(endDate)
                    .alarmName(alarmName)
                    .historyItemType(HistoryItemType.ACTION)
                    .build();

                // Use the paginator to paginate through alarm history pages.
                DescribeAlarmHistoryPublisher historyPublisher = 
getAsyncClient().describeAlarmHistoryPaginator(historyRequest);
                CompletableFuture<Void> future = historyPublisher
                    .subscribe(response -> 
response.alarmHistoryItems().forEach(item -> {
                        logger.info("History summary: {}", 
item.historySummary());
                        logger.info("Timestamp: {}", item.timestamp());
                    }))
                    .whenComplete((result, exception) -> {
                        if (exception != null) {
```

```
                            logger.error("Error occurred while getting alarm 
history: " + exception.getMessage(), exception);
                        } else {
                            logger.info("Successfully retrieved all alarm 
history.");
                        }
                    });

                // Return the future to the calling code for further handling
                return future;
            } catch (Exception e) {
                throw new RuntimeException("Failed to process alarm history", e);
            }
        }).whenComplete((result, exception) -> {
            if (exception != null) {
                throw new RuntimeException("Error completing alarm history 
processing", exception);
            }
        });
    }



    /**
     * Checks for a metric alarm in AWS CloudWatch.
     *
     * @param fileName the name of the file containing the JSON configuration for 
the custom metric
     * @return a {@link CompletableFuture} that completes when the check for the 
metric alarm is complete
     */
    public CompletableFuture<Void> checkForMetricAlarmAsync(String fileName) {
        CompletableFuture<String> readFileFuture = 
CompletableFuture.supplyAsync(() -> {
            try {
                JsonParser parser = new JsonFactory().createParser(new 
File(fileName));
                com.fasterxml.jackson.databind.JsonNode rootNode = new 
ObjectMapper().readTree(parser);
                return rootNode.toString(); // Return JSON as a string for 
further processing
            } catch (IOException e) {
                throw new RuntimeException("Failed to read file", e);
            }
```

```
        });

        return readFileFuture.thenCompose(jsonContent -> {
            try {
                com.fasterxml.jackson.databind.JsonNode rootNode = new
 ObjectMapper().readTree(jsonContent);
                String customMetricNamespace =
 rootNode.findValue("customMetricNamespace").asText();
                String customMetricName =
 rootNode.findValue("customMetricName").asText();

                DescribeAlarmsForMetricRequest metricRequest =
 DescribeAlarmsForMetricRequest.builder()
                    .metricName(customMetricName)
                    .namespace(customMetricNamespace)
                    .build();

                return checkForAlarmAsync(metricRequest, customMetricName, 10);

            } catch (IOException e) {
                throw new RuntimeException("Failed to parse JSON content", e);
            }
        }).whenComplete((result, exception) -> {
            if (exception != null) {
                throw new RuntimeException("Error checking metric alarm",
 exception);
            }
        });
    }

    // Recursive method to check for the alarm.

    /**
     * Checks for the existence of an alarm asynchronously for the specified
 metric.
     *
     * @param metricRequest    the request to describe the alarms for the
 specified metric
     * @param customMetricName the name of the custom metric to check for an
 alarm
     * @param retries          the number of retries to perform if no alarm is
 found
     * @return a {@link CompletableFuture} that completes when an alarm is found
 or the maximum number of retries has been reached
```

```
     */
    private static CompletableFuture<Void>
checkForAlarmAsync(DescribeAlarmsForMetricRequest metricRequest, String
customMetricName, int retries) {
        if (retries == 0) {
            return CompletableFuture.completedFuture(null).thenRun(() ->
                logger.info("No Alarm state found for {} after 10 retries.",
customMetricName)
            );
        }

        return
(getAsyncClient().describeAlarmsForMetric(metricRequest).thenCompose(response ->
{
            if (response.hasMetricAlarms()) {
                logger.info("Alarm state found for {}", customMetricName);
                return CompletableFuture.completedFuture(null); // Alarm found,
complete the future
            } else {
                return CompletableFuture.runAsync(() -> {
                    try {
                        Thread.sleep(20000);
                        logger.info(".");
                    } catch (InterruptedException e) {
                        throw new RuntimeException("Interrupted while waiting to
retry", e);
                    }
                }).thenCompose(v -> checkForAlarmAsync(metricRequest,
customMetricName, retries - 1)); // Recursive call
            }
        }));
    }


    /**
     * Adds metric data for an alarm asynchronously.
     *
     * @param fileName the name of the JSON file containing the metric data
     * @return a CompletableFuture that asynchronously returns the
PutMetricDataResponse
     */
    public CompletableFuture<PutMetricDataResponse>
addMetricDataForAlarmAsync(String fileName) {
```

```
        CompletableFuture<String> readFileFuture =
 CompletableFuture.supplyAsync(() -> {
            try {
                JsonParser parser = new JsonFactory().createParser(new
 File(fileName));
                com.fasterxml.jackson.databind.JsonNode rootNode = new
 ObjectMapper().readTree(parser);
                return rootNode.toString(); // Return JSON as a string for
 further processing
            } catch (IOException e) {
                throw new RuntimeException("Failed to read file", e);
            }
        });

        return readFileFuture.thenCompose(jsonContent -> {
            try {
                com.fasterxml.jackson.databind.JsonNode rootNode = new
 ObjectMapper().readTree(jsonContent);
                String customMetricNamespace =
 rootNode.findValue("customMetricNamespace").asText();
                String customMetricName =
 rootNode.findValue("customMetricName").asText();
                Instant instant = Instant.now();

                // Create MetricDatum objects.
                MetricDatum datum1 = MetricDatum.builder()
                    .metricName(customMetricName)
                    .unit(StandardUnit.NONE)
                    .value(1001.00)
                    .timestamp(instant)
                    .build();

                MetricDatum datum2 = MetricDatum.builder()
                    .metricName(customMetricName)
                    .unit(StandardUnit.NONE)
                    .value(1002.00)
                    .timestamp(instant)
                    .build();

                List<MetricDatum> metricDataList = new ArrayList<>();
                metricDataList.add(datum1);
                metricDataList.add(datum2);

                // Build the PutMetricData request.
```

```
                PutMetricDataRequest request = PutMetricDataRequest.builder()
                    .namespace(customMetricNamespace)
                    .metricData(metricDataList)
                    .build();

                // Send the request asynchronously.
                return getAsyncClient().putMetricData(request);

            } catch (IOException e) {
                CompletableFuture<PutMetricDataResponse> failedFuture = new
 CompletableFuture<>();
                failedFuture.completeExceptionally(new RuntimeException("Failed
 to parse JSON content", e));
                return failedFuture;
            }
        }).whenComplete((response, exception) -> {
            if (exception != null) {
                logger.error("Failed to put metric data: " +
 exception.getMessage(), exception);
            } else {
                logger.info("Added metric values for metric.");
            }
        });
    }


    /**
     * Retrieves custom metric data from the AWS CloudWatch service.
     *
     * @param fileName the name of the file containing the custom metric
 information
     * @return a {@link CompletableFuture} that completes when the metric data
 has been retrieved
     */
    public CompletableFuture<Void> getCustomMetricDataAsync(String fileName) {
        CompletableFuture<String> readFileFuture =
 CompletableFuture.supplyAsync(() -> {
            try {
                // Read values from the JSON file.
                JsonParser parser = new JsonFactory().createParser(new
 File(fileName));
                com.fasterxml.jackson.databind.JsonNode rootNode = new
 ObjectMapper().readTree(parser);
```

```
                return rootNode.toString(); // Return JSON as a string for
further processing
            } catch (IOException e) {
                throw new RuntimeException("Failed to read file", e);
            }
        });

        return readFileFuture.thenCompose(jsonContent -> {
            try {
                // Parse the JSON string to extract relevant values.
                com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(jsonContent);
                String customMetricNamespace =
rootNode.findValue("customMetricNamespace").asText();
                String customMetricName =
rootNode.findValue("customMetricName").asText();

                // Set the current time and date range for metric query.
                Instant nowDate = Instant.now();
                long hours = 1;
                long minutes = 30;
                Instant endTime = nowDate.plus(hours,
ChronoUnit.HOURS).plus(minutes, ChronoUnit.MINUTES);

                Metric met = Metric.builder()
                    .metricName(customMetricName)
                    .namespace(customMetricNamespace)
                    .build();

                MetricStat metStat = MetricStat.builder()
                    .stat("Maximum")
                    .period(60)  // Assuming period in seconds
                    .metric(met)
                    .build();

                MetricDataQuery dataQuery = MetricDataQuery.builder()
                    .metricStat(metStat)
                    .id("foo2")
                    .returnData(true)
                    .build();

                List<MetricDataQuery> dq = new ArrayList<>();
                dq.add(dataQuery);
```

```
                GetMetricDataRequest getMetricDataRequest =
GetMetricDataRequest.builder()
                    .maxDatapoints(10)
                    .scanBy(ScanBy.TIMESTAMP_DESCENDING)
                    .startTime(nowDate)
                    .endTime(endTime)
                    .metricDataQueries(dq)
                    .build();

                // Call the async method for CloudWatch data retrieval.
                return getAsyncClient().getMetricData(getMetricDataRequest);

            } catch (IOException e) {
                throw new RuntimeException("Failed to parse JSON content", e);
            }
        }).thenAccept(response -> {
            List<MetricDataResult> data = response.metricDataResults();
            for (MetricDataResult item : data) {
                logger.info("The label is: {}", item.label());
                logger.info("The status code is: {}",
item.statusCode().toString());
            }
        }).exceptionally(exception -> {
            throw new RuntimeException("Failed to get metric data", exception);
        });
    }


    /**
     * Describes the CloudWatch alarms of the 'METRIC_ALARM' type.
     *
     * @return a {@link CompletableFuture} that represents the asynchronous
operation
     * of describing the CloudWatch alarms. The future completes when the
     * operation is finished, either successfully or with an error.
     */
    public CompletableFuture<Void> describeAlarmsAsync() {
        List<AlarmType> typeList = new ArrayList<>();
        typeList.add(AlarmType.METRIC_ALARM);
        DescribeAlarmsRequest alarmsRequest = DescribeAlarmsRequest.builder()
            .alarmTypes(typeList)
            .maxRecords(10)
            .build();
```

```
        return getAsyncClient().describeAlarms(alarmsRequest)
            .thenAccept(response -> {
                List<MetricAlarm> alarmList = response.metricAlarms();
                for (MetricAlarm alarm : alarmList) {
                    logger.info("Alarm name: {}", alarm.alarmName());
                    logger.info("Alarm description: {} ",
alarm.alarmDescription());
                }
            })
            .whenComplete((response, ex) -> {
                if (ex != null) {
                    logger.info("Failed to describe alarms: {}",
ex.getMessage());
                } else {
                    logger.info("Successfully described alarms.");
                }
            });
    }

    /**
     * Creates an alarm based on the configuration provided in a JSON file.
     *
     * @param fileName the name of the JSON file containing the alarm
configuration
     * @return a CompletableFuture that represents the asynchronous operation of
creating the alarm
     * @throws RuntimeException if an exception occurs while reading the JSON
file or creating the alarm
     */
    public CompletableFuture<String> createAlarmAsync(String fileName) {
        com.fasterxml.jackson.databind.JsonNode rootNode;
        try {
            JsonParser parser = new JsonFactory().createParser(new
File(fileName));
            rootNode = new ObjectMapper().readTree(parser);
        } catch (IOException e) {
            throw new RuntimeException("Failed to read the alarm configuration
file", e);
        }

        // Extract values from the JSON node.
        String customMetricNamespace =
rootNode.findValue("customMetricNamespace").asText();
```

```
        String customMetricName =
rootNode.findValue("customMetricName").asText();
        String alarmName = rootNode.findValue("exampleAlarmName").asText();
        String emailTopic = rootNode.findValue("emailTopic").asText();
        String accountId = rootNode.findValue("accountId").asText();
        String region = rootNode.findValue("region").asText();

        // Create a List for alarm actions.
        List<String> alarmActions = new ArrayList<>();
        alarmActions.add("arn:aws:sns:" + region + ":" + accountId + ":" +
emailTopic);

        PutMetricAlarmRequest alarmRequest = PutMetricAlarmRequest.builder()
            .alarmActions(alarmActions)
            .alarmDescription("Example metric alarm")
            .alarmName(alarmName)

.comparisonOperator(ComparisonOperator.GREATER_THAN_OR_EQUAL_TO_THRESHOLD)
            .threshold(100.00)
            .metricName(customMetricName)
            .namespace(customMetricNamespace)
            .evaluationPeriods(1)
            .period(10)
            .statistic("Maximum")
            .datapointsToAlarm(1)
            .treatMissingData("ignore")
            .build();

        // Call the putMetricAlarm asynchronously and handle the result.
        return getAsyncClient().putMetricAlarm(alarmRequest)
            .handle((response, ex) -> {
                if (ex != null) {
                    logger.info("Failed to create alarm: {}", ex.getMessage());
                    throw new RuntimeException("Failed to create alarm", ex);
                } else {
                    logger.info("{} was successfully created!", alarmName);
                    return alarmName;
                }
            });
    }

    /**
     * Adds a metric to a dashboard asynchronously.
     *
```

```
     * @param fileName      the name of the file containing the dashboard content
     * @param dashboardName the name of the dashboard to be updated
     * @return a {@link CompletableFuture} representing the asynchronous
 operation, which will complete with a
     * {@link PutDashboardResponse} when the dashboard is successfully updated
     */
    public CompletableFuture<PutDashboardResponse>
 addMetricToDashboardAsync(String fileName, String dashboardName) {
        String dashboardBody;
        try {
            dashboardBody = readFileAsString(fileName);
        } catch (IOException e) {
            throw new RuntimeException("Failed to read the dashboard file", e);
        }

        PutDashboardRequest dashboardRequest = PutDashboardRequest.builder()
            .dashboardName(dashboardName)
            .dashboardBody(dashboardBody)
            .build();

        return getAsyncClient().putDashboard(dashboardRequest)
            .handle((response, ex) -> {
                if (ex != null) {
                    logger.info("Failed to update dashboard: {}",
 ex.getMessage());
                    throw new RuntimeException("Error updating dashboard", ex);
                } else {
                    logger.info("{} was successfully updated.", dashboardName);
                    return response;
                }
            });
    }

    /**
     * Creates a new custom metric.
     *
     * @param dataPoint the data point to be added to the custom metric
     * @return a {@link CompletableFuture} representing the asynchronous
 operation of adding the custom metric
     */
    public CompletableFuture<PutMetricDataResponse>
 createNewCustomMetricAsync(Double dataPoint) {
        Dimension dimension = Dimension.builder()
            .name("UNIQUE_PAGES")
```

```
            .value("URLS")
            .build();

        // Set an Instant object for the current time in UTC.
        String time =
ZonedDateTime.now(ZoneOffset.UTC).format(DateTimeFormatter.ISO_INSTANT);
        Instant instant = Instant.parse(time);

        // Create the MetricDatum.
        MetricDatum datum = MetricDatum.builder()
            .metricName("PAGES_VISITED")
            .unit(StandardUnit.NONE)
            .value(dataPoint)
            .timestamp(instant)
            .dimensions(dimension)
            .build();

        PutMetricDataRequest request = PutMetricDataRequest.builder()
            .namespace("SITE/TRAFFIC")
            .metricData(datum)
            .build();

        return getAsyncClient().putMetricData(request)
            .whenComplete((response, ex) -> {
                if (ex != null) {
                    throw new RuntimeException("Error adding custom metric", ex);
                } else {
                    logger.info("Successfully added metric values for
PAGES_VISITED.");
                }
            });
    }

    /**
     * Lists the available dashboards.
     *
     * @return a {@link CompletableFuture} that completes when the operation is
finished.
     * The future will complete exceptionally if an error occurs while listing
the dashboards.
     */
    public CompletableFuture<Void> listDashboardsAsync() {
        ListDashboardsRequest listDashboardsRequest =
ListDashboardsRequest.builder().build();
```

```
        ListDashboardsPublisher paginator =
getAsyncClient().listDashboardsPaginator(listDashboardsRequest);
        return paginator.subscribe(response -> {
            response.dashboardEntries().forEach(entry -> {
                logger.info("Dashboard name is: {} ", entry.dashboardName());
                logger.info("Dashboard ARN is: {} ", entry.dashboardArn());
            });
        }).exceptionally(ex -> {
            logger.info("Failed to list dashboards: {} ", ex.getMessage());
            throw new RuntimeException("Error occurred while listing dashboards",
ex);
        });
    }


    /**
     * Creates a new dashboard with the specified name and metrics from the given
file.
     *
     * @param dashboardName the name of the dashboard to be created
     * @param fileName      the name of the file containing the dashboard body
     * @return a {@link CompletableFuture} representing the asynchronous
operation of creating the dashboard
     * @throws IOException if there is an error reading the dashboard body from
the file
     */
    public CompletableFuture<PutDashboardResponse>
createDashboardWithMetricsAsync(String dashboardName, String fileName) throws
IOException {
        String dashboardBody = readFileAsString(fileName);
        PutDashboardRequest dashboardRequest = PutDashboardRequest.builder()
            .dashboardName(dashboardName)
            .dashboardBody(dashboardBody)
            .build();

        return getAsyncClient().putDashboard(dashboardRequest)
            .handle((response, ex) -> {
                if (ex != null) {
                    logger.info("Failed to create dashboard: {}",
ex.getMessage());
                    throw new RuntimeException("Dashboard creation failed", ex);
                } else {
                    // Handle the normal response case
                    logger.info("{} was successfully created.", dashboardName);
```

```
                    List<DashboardValidationMessage> messages =
response.dashboardValidationMessages();
                    if (messages.isEmpty()) {
                        logger.info("There are no messages in the new
Dashboard.");
                    } else {
                        for (DashboardValidationMessage message : messages) {
                            logger.info("Message: {}", message.message());
                        }
                    }
                    return response; // Return the response for further use
                }
            });
    }


    /**
     * Retrieves the metric statistics for the "EstimatedCharges" metric in the
"AWS/Billing" namespace.
     *
     * @param costDateWeek the start date for the metric statistics, in the
format of an ISO-8601 date string (e.g., "2023-04-05")
     * @return a {@link CompletableFuture} that, when completed, contains the
{@link GetMetricStatisticsResponse} with the retrieved metric statistics
     * @throws RuntimeException if the metric statistics cannot be retrieved
successfully
     */
    public CompletableFuture<GetMetricStatisticsResponse>
getMetricStatisticsAsync(String costDateWeek) {
        Instant start = Instant.parse(costDateWeek);
        Instant endDate = Instant.now();

        // Define dimension
        Dimension dimension = Dimension.builder()
            .name("Currency")
            .value("USD")
            .build();

        List<Dimension> dimensionList = new ArrayList<>();
        dimensionList.add(dimension);

        GetMetricStatisticsRequest statisticsRequest =
GetMetricStatisticsRequest.builder()
            .metricName("EstimatedCharges")
```

```
            .namespace("AWS/Billing")
            .dimensions(dimensionList)
            .statistics(Statistic.MAXIMUM)
            .startTime(start)
            .endTime(endDate)
            .period(86400) // One day period
            .build();

        return getAsyncClient().getMetricStatistics(statisticsRequest)
            .whenComplete((response, exception) -> {
                if (response != null) {
                    List<Datapoint> data = response.datapoints();
                    if (!data.isEmpty()) {
                        for (Datapoint datapoint : data) {
                            logger.info("Timestamp: {} Maximum value: {})",
 datapoint.timestamp(), datapoint.maximum());
                        }
                    } else {
                        logger.info("The returned data list is empty");
                    }
                } else {
                    throw new RuntimeException("Failed to get metric statistics:
 " + exception.getMessage(), exception);
                }
            });
    }


    /**
     * Retrieves and displays metric statistics for the specified parameters.
     *
     * @param nameSpace    the namespace for the metric
     * @param metVal       the name of the metric
     * @param metricOption the statistic to retrieve for the metric (e.g.,
 "Maximum", "Average")
     * @param date         the date for which to retrieve the metric statistics,
 in the format "yyyy-MM-dd'T'HH:mm:ss'Z'"
     * @param myDimension  the dimension(s) to filter the metric statistics by
     * @return a {@link CompletableFuture} that completes when the metric
 statistics have been retrieved and displayed
     */
    public CompletableFuture<GetMetricStatisticsResponse>
 getAndDisplayMetricStatisticsAsync(String nameSpace, String metVal,
```

```
            String metricOption, String date, Dimension myDimension) {

        Instant start = Instant.parse(date);
        Instant endDate = Instant.now();

        // Building the request for metric statistics.
        GetMetricStatisticsRequest statisticsRequest =
GetMetricStatisticsRequest.builder()
            .endTime(endDate)
            .startTime(start)
            .dimensions(myDimension)
            .metricName(metVal)
            .namespace(nameSpace)
            .period(86400) // 1 day period
            .statistics(Statistic.fromValue(metricOption))
            .build();

        return getAsyncClient().getMetricStatistics(statisticsRequest)
            .whenComplete((response, exception) -> {
                if (response != null) {
                    List<Datapoint> data = response.datapoints();
                    if (!data.isEmpty()) {
                        for (Datapoint datapoint : data) {
                            logger.info("Timestamp: {} Maximum value: {}",
datapoint.timestamp(), datapoint.maximum());
                        }
                    } else {
                        logger.info("The returned data list is empty");
                    }
                } else {
                    logger.info("Failed to get metric statistics: {} ",
exception.getMessage());
                }
            })
            .exceptionally(exception -> {
                throw new RuntimeException("Error while getting metric
statistics: " + exception.getMessage(), exception);
            });
    }


    /**
     * Retrieves a list of metric names for the specified namespace.
```

```
     *
     * @param namespace the namespace for which to retrieve the metric names
     * @return a {@link CompletableFuture} that, when completed, contains an
 {@link ArrayList} of
     * the metric names in the specified namespace
     * @throws RuntimeException if an error occurs while listing the metrics
     */
    public CompletableFuture<ArrayList<String>> listMetsAsync(String namespace) {
        ListMetricsRequest request = ListMetricsRequest.builder()
            .namespace(namespace)
            .build();

        ListMetricsPublisher metricsPaginator =
 getAsyncClient().listMetricsPaginator(request);
        Set<String> metSet = new HashSet<>();
        CompletableFuture<Void> future = metricsPaginator.subscribe(response -> {
            response.metrics().forEach(metric -> {
                String metricName = metric.metricName();
                metSet.add(metricName);
            });
        });

        return future
            .thenApply(ignored -> new ArrayList<>(metSet))
            .exceptionally(exception -> {
                throw new RuntimeException("Failed to list metrics: " +
 exception.getMessage(), exception);
            });
    }

    /**
     * Lists the available namespaces for the current AWS account.
     *
     * @return a {@link CompletableFuture} that, when completed, contains an
 {@link ArrayList} of the available namespace names.
     * @throws RuntimeException if an error occurs while listing the namespaces.
     */
    public CompletableFuture<ArrayList<String>> listNameSpacesAsync() {
        ArrayList<String> nameSpaceList = new ArrayList<>();
        ListMetricsRequest request = ListMetricsRequest.builder().build();

        ListMetricsPublisher metricsPaginator =
 getAsyncClient().listMetricsPaginator(request);
        CompletableFuture<Void> future = metricsPaginator.subscribe(response -> {
```

```
            response.metrics().forEach(metric -> {
                String namespace = metric.namespace();
                if (!nameSpaceList.contains(namespace)) {
                    nameSpaceList.add(namespace);
                }
            });
        });

        return future
            .thenApply(ignored -> nameSpaceList)
            .exceptionally(exception -> {
                throw new RuntimeException("Failed to list namespaces: " +
exception.getMessage(), exception);
            });
    }
    /**
     * Retrieves the specific metric asynchronously.
     *
     * @param namespace the namespace of the metric to retrieve
     * @return a CompletableFuture that completes with the first dimension of the
first metric found in the specified namespace,
     * or throws a RuntimeException if an error occurs or no metrics or
dimensions are found
     */
    public CompletableFuture<Dimension> getSpecificMetAsync(String namespace) {
        ListMetricsRequest request = ListMetricsRequest.builder()
            .namespace(namespace)
            .build();

        return getAsyncClient().listMetrics(request).handle((response, exception)
-> {
            if (exception != null) {
                logger.info("Error occurred while listing metrics: {} ",
exception.getMessage());
                throw new RuntimeException("Failed to retrieve specific metric
dimension", exception);
            } else {
                List<Metric> myList = response.metrics();
                if (!myList.isEmpty()) {
                    Metric metric = myList.get(0);
                    if (!metric.dimensions().isEmpty()) {
                        return metric.dimensions().get(0); // Return the first
dimension
                    }
```

```
                }
                throw new RuntimeException("No metrics or dimensions found");
            }
        });
    }

    public static String readFileAsString(String file) throws IOException {
        return new String(Files.readAllBytes(Paths.get(file)));
    }
}
```

- For API details, see the following topics in *Amazon SDK for Java 2.x API Reference*.
  - DeleteAlarms
  - DeleteAnomalyDetector
  - DeleteDashboards
  - DescribeAlarmHistory
  - DescribeAlarms
  - DescribeAlarmsForMetric
  - DescribeAnomalyDetectors
  - GetMetricData
  - GetMetricStatistics
  - GetMetricWidgetImage
  - ListMetrics
  - PutAnomalyDetector
  - PutDashboard
  - PutMetricAlarm
  - PutMetricData

Kotlin

SDK for Kotlin

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [Amazon Code Examples Repository](https://github.com/awsdocs/aws-doc-sdk-examples/tree/main/kotlin/services/cloudwatch#code-examples).

Run an interactive scenario demonstrating CloudWatch features.

```
/**
 Before running this Kotlin code example, set up your development environment,
 including your credentials.

 For more information, see the following documentation topic:
 https://docs.aws.amazon.com/sdk-for-kotlin/latest/developer-guide/setup.html

 To enable billing metrics and statistics for this example, make sure billing
 alerts are enabled for your account:
 https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/
monitor_estimated_charges_with_cloudwatch.html#turning_on_billing_metrics

 This Kotlin code example performs the following tasks:

 1. List available namespaces from Amazon CloudWatch. Select a namespace from the
 list.
 2. List available metrics within the selected namespace.
 3. Get statistics for the selected metric over the last day.
 4. Get CloudWatch estimated billing for the last week.
 5. Create a new CloudWatch dashboard with metrics.
 6. List dashboards using a paginator.
 7. Create a new custom metric by adding data for it.
 8. Add the custom metric to the dashboard.
 9. Create an alarm for the custom metric.
 10. Describe current alarms.
 11. Get current data for the new custom metric.
 12. Push data into the custom metric to trigger the alarm.
 13. Check the alarm state using the action DescribeAlarmsForMetric.
 14. Get alarm history for the new alarm.
 15. Add an anomaly detector for the custom metric.
```

```
 16. Describe current anomaly detectors.
 17. Get a metric image for the custom metric.
 18. Clean up the Amazon CloudWatch resources.
 */

val DASHES: String? = String(CharArray(80)).replace("\u0000", "-")

suspend fun main(args: Array<String>) {
    val usage = """
        Usage:
            <myDate> <costDateWeek> <dashboardName> <dashboardJson>
 <dashboardAdd> <settings> <metricImage>

        Where:
            myDate - The start date to use to get metric statistics. (For
 example, 2023-01-11T18:35:24.00Z.)
            costDateWeek - The start date to use to get AWS Billing and Cost
 Management statistics. (For example, 2023-01-11T18:35:24.00Z.)
            dashboardName - The name of the dashboard to create.
            dashboardJson - The location of a JSON file to use to create a
 dashboard. (See Readme file.)
            dashboardAdd - The location of a JSON file to use to update a
 dashboard. (See Readme file.)
            settings - The location of a JSON file from which various values are
 read. (See Readme file.)
            metricImage - The location of a BMP file that is used to create a
 graph.
    """

    if (args.size != 7) {
        println(usage)
        System.exit(1)
    }

    val myDate = args[0]
    val costDateWeek = args[1]
    val dashboardName = args[2]
    val dashboardJson = args[3]
    val dashboardAdd = args[4]
    val settings = args[5]
    var metricImage = args[6]
    val dataPoint = "10.0".toDouble()
    val inOb = Scanner(System.`in`)
```

```
    println(DASHES)
    println("Welcome to the Amazon CloudWatch example scenario.")
    println(DASHES)

    println(DASHES)
    println("1. List at least five available unique namespaces from Amazon
CloudWatch. Select a CloudWatch namespace from the list.")
    val list: ArrayList<String> = listNameSpaces()
    for (z in 0..4) {
        println("    ${z + 1}. ${list[z]}")
    }

    var selectedNamespace: String
    var selectedMetrics = ""
    var num = inOb.nextLine().toInt()
    println("You selected $num")

    if (1 <= num && num <= 5) {
        selectedNamespace = list[num - 1]
    } else {
        println("You did not select a valid option.")
        exitProcess(1)
    }
    println("You selected $selectedNamespace")
    println(DASHES)

    println(DASHES)
    println("2. List available metrics within the selected namespace and select
one from the list.")
    val metList = listMets(selectedNamespace)
    for (z in 0..4) {
        println("    ${ z + 1}. ${metList?.get(z)}")
    }
    num = inOb.nextLine().toInt()
    if (1 <= num && num <= 5) {
        selectedMetrics = metList!![num - 1]
    } else {
        println("You did not select a valid option.")
        System.exit(1)
    }
    println("You selected $selectedMetrics")
    val myDimension = getSpecificMet(selectedNamespace)
    if (myDimension == null) {
        println("Error - Dimension is null")
```

```
        exitProcess(1)
    }
    println(DASHES)

    println(DASHES)
    println("3. Get statistics for the selected metric over the last day.")
    val metricOption: String
    val statTypes = ArrayList<String>()
    statTypes.add("SampleCount")
    statTypes.add("Average")
    statTypes.add("Sum")
    statTypes.add("Minimum")
    statTypes.add("Maximum")

    for (t in 0..4) {
        println("    ${t + 1}. ${statTypes[t]}")
    }
    println("Select a metric statistic by entering a number from the preceding
 list:")
    num = inOb.nextLine().toInt()
    if (1 <= num && num <= 5) {
        metricOption = statTypes[num - 1]
    } else {
        println("You did not select a valid option.")
        exitProcess(1)
    }
    println("You selected $metricOption")
    getAndDisplayMetricStatistics(selectedNamespace, selectedMetrics,
 metricOption, myDate, myDimension)
    println(DASHES)

    println(DASHES)
    println("4. Get CloudWatch estimated billing for the last week.")
    getMetricStatistics(costDateWeek)
    println(DASHES)

    println(DASHES)
    println("5. Create a new CloudWatch dashboard with metrics.")
    createDashboardWithMetrics(dashboardName, dashboardJson)
    println(DASHES)

    println(DASHES)
    println("6. List dashboards using a paginator.")
    listDashboards()
```

```
    println(DASHES)

    println(DASHES)
    println("7. Create a new custom metric by adding data to it.")
    createNewCustomMetric(dataPoint)
    println(DASHES)

    println(DASHES)
    println("8. Add an additional metric to the dashboard.")
    addMetricToDashboard(dashboardAdd, dashboardName)
    println(DASHES)

    println(DASHES)
    println("9. Create an alarm for the custom metric.")
    val alarmName: String = createAlarm(settings)
    println(DASHES)

    println(DASHES)
    println("10. Describe 10 current alarms.")
    describeAlarms()
    println(DASHES)

    println(DASHES)
    println("11. Get current data for the new custom metric.")
    getCustomMetricData(settings)
    println(DASHES)

    println(DASHES)
    println("12. Push data into the custom metric to trigger the alarm.")
    addMetricDataForAlarm(settings)
    println(DASHES)

    println(DASHES)
    println("13. Check the alarm state using the action
 DescribeAlarmsForMetric.")
    checkForMetricAlarm(settings)
    println(DASHES)

    println(DASHES)
    println("14. Get alarm history for the new alarm.")
    getAlarmHistory(settings, myDate)
    println(DASHES)

    println(DASHES)
```

```
    println("15. Add an anomaly detector for the custom metric.")
    addAnomalyDetector(settings)
    println(DASHES)

    println(DASHES)
    println("16. Describe current anomaly detectors.")
    describeAnomalyDetectors(settings)
    println(DASHES)

    println(DASHES)
    println("17. Get a metric image for the custom metric.")
    getAndOpenMetricImage(metricImage)
    println(DASHES)

    println(DASHES)
    println("18. Clean up the Amazon CloudWatch resources.")
    deleteDashboard(dashboardName)
    deleteAlarm(alarmName)
    deleteAnomalyDetector(settings)
    println(DASHES)

    println(DASHES)
    println("The Amazon CloudWatch example scenario is complete.")
    println(DASHES)
}

suspend fun deleteAnomalyDetector(fileName: String) {
    // Read values from the JSON file.
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val customMetricNamespace =
 rootNode.findValue("customMetricNamespace").asText()
    val customMetricName = rootNode.findValue("customMetricName").asText()

    val singleMetricAnomalyDetectorVal =
        SingleMetricAnomalyDetector {
            metricName = customMetricName
            namespace = customMetricNamespace
            stat = "Maximum"
        }

    val request =
        DeleteAnomalyDetectorRequest {
            singleMetricAnomalyDetector = singleMetricAnomalyDetectorVal
```

```
        }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient.deleteAnomalyDetector(request)
        println("Successfully deleted the Anomaly Detector.")
    }
}

suspend fun deleteAlarm(alarmNameVal: String) {
    val request =
        DeleteAlarmsRequest {
            alarmNames = listOf(alarmNameVal)
        }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient.deleteAlarms(request)
        println("Successfully deleted alarm $alarmNameVal")
    }
}

suspend fun deleteDashboard(dashboardName: String) {
    val dashboardsRequest =
        DeleteDashboardsRequest {
            dashboardNames = listOf(dashboardName)
        }
    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient.deleteDashboards(dashboardsRequest)
        println("$dashboardName was successfully deleted.")
    }
}

suspend fun getAndOpenMetricImage(fileName: String) {
    println("Getting Image data for custom metric.")
    val myJSON = """{
        "title": "Example Metric Graph",
        "view": "timeSeries",
        "stacked ": false,
        "period": 10,
        "width": 1400,
        "height": 600,
        "metrics": [
            [
            "AWS/Billing",
            "EstimatedCharges",
```

```
            "Currency",
            "USD"
            ]
        ]
        }"""

    val imageRequest =
        GetMetricWidgetImageRequest {
            metricWidget = myJSON
        }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        val response = cwClient.getMetricWidgetImage(imageRequest)
        val bytes = response.metricWidgetImage
        if (bytes != null) {
            File(fileName).writeBytes(bytes)
        }
    }
    println("You have successfully written data to $fileName")
}

suspend fun describeAnomalyDetectors(fileName: String) {
    // Read values from the JSON file.
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val customMetricNamespace =
 rootNode.findValue("customMetricNamespace").asText()
    val customMetricName = rootNode.findValue("customMetricName").asText()

    val detectorsRequest =
        DescribeAnomalyDetectorsRequest {
            maxResults = 10
            metricName = customMetricName
            namespace = customMetricNamespace
        }
    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        val response = cwClient.describeAnomalyDetectors(detectorsRequest)
        response.anomalyDetectors?.forEach { detector ->
            println("Metric name:
 ${detector.singleMetricAnomalyDetector?.metricName}")
            println("State: ${detector.stateValue}")
        }
    }
}
```

```
suspend fun addAnomalyDetector(fileName: String?) {
    // Read values from the JSON file.
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val customMetricNamespace =
 rootNode.findValue("customMetricNamespace").asText()
    val customMetricName = rootNode.findValue("customMetricName").asText()

    val singleMetricAnomalyDetectorVal =
        SingleMetricAnomalyDetector {
            metricName = customMetricName
            namespace = customMetricNamespace
            stat = "Maximum"
        }

    val anomalyDetectorRequest =
        PutAnomalyDetectorRequest {
            singleMetricAnomalyDetector = singleMetricAnomalyDetectorVal
        }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient.putAnomalyDetector(anomalyDetectorRequest)
        println("Added anomaly detector for metric $customMetricName.")
    }
}

suspend fun getAlarmHistory(
    fileName: String,
    date: String,
) {
    // Read values from the JSON file.
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val alarmNameVal = rootNode.findValue("exampleAlarmName").asText()
    val start = Instant.parse(date)
    val endDateVal = Instant.now()

    val historyRequest =
        DescribeAlarmHistoryRequest {
            startDate =
                aws.smithy.kotlin.runtime.time
                    .Instant(start)
            endDate =
```

```
                aws.smithy.kotlin.runtime.time
                    .Instant(endDateVal)
            alarmName = alarmNameVal
            historyItemType = HistoryItemType.Action
        }

    CloudWatchClient {
        credentialsProvider = EnvironmentCredentialsProvider()
        region = "us-east-1"
    }.use { cwClient ->
        val response = cwClient.describeAlarmHistory(historyRequest)
        val historyItems = response.alarmHistoryItems
        if (historyItems != null) {
            if (historyItems.isEmpty()) {
                println("No alarm history data found for $alarmNameVal.")
            } else {
                for (item in historyItems) {
                    println("History summary ${item.historySummary}")
                    println("Time stamp: ${item.timestamp}")
                }
            }
        }
    }
}

suspend fun checkForMetricAlarm(fileName: String?) {
    // Read values from the JSON file.
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val customMetricNamespace =
 rootNode.findValue("customMetricNamespace").asText()
    val customMetricName = rootNode.findValue("customMetricName").asText()
    var hasAlarm = false
    var retries = 10

    val metricRequest =
        DescribeAlarmsForMetricRequest {
            metricName = customMetricName
            namespace = customMetricNamespace
        }
    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        while (!hasAlarm && retries > 0) {
            val response = cwClient.describeAlarmsForMetric(metricRequest)
            if (response.metricAlarms?.count()!! > 0) {
```

```
                hasAlarm = true
            }
            retries--
            delay(20000)
            println(".")
        }
        if (!hasAlarm) {
            println("No Alarm state found for $customMetricName after 10
 retries.")
        } else {
            println("Alarm state found for $customMetricName.")
        }
    }
}

suspend fun addMetricDataForAlarm(fileName: String?) {
    // Read values from the JSON file.
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val customMetricNamespace =
 rootNode.findValue("customMetricNamespace").asText()
    val customMetricName = rootNode.findValue("customMetricName").asText()

    // Set an Instant object.
    val time =
 ZonedDateTime.now(ZoneOffset.UTC).format(DateTimeFormatter.ISO_INSTANT)
    val instant = Instant.parse(time)
    val datum =
        MetricDatum {
            metricName = customMetricName
            unit = StandardUnit.None
            value = 1001.00
            timestamp =
                aws.smithy.kotlin.runtime.time
                    .Instant(instant)
        }

    val datum2 =
        MetricDatum {
            metricName = customMetricName
            unit = StandardUnit.None
            value = 1002.00
            timestamp =
                aws.smithy.kotlin.runtime.time
```

```
                    .Instant(instant)
        }

    val metricDataList = ArrayList<MetricDatum>()
    metricDataList.add(datum)
    metricDataList.add(datum2)

    val request =
        PutMetricDataRequest {
            namespace = customMetricNamespace
            metricData = metricDataList
        }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient.putMetricData(request)
        println("Added metric values for for metric $customMetricName")
    }
}

suspend fun getCustomMetricData(fileName: String) {
    // Read values from the JSON file.
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val customMetricNamespace =
 rootNode.findValue("customMetricNamespace").asText()
    val customMetricName = rootNode.findValue("customMetricName").asText()

    // Set the date.
    val nowDate = Instant.now()
    val hours: Long = 1
    val minutes: Long = 30
    val date2 =
        nowDate.plus(hours, ChronoUnit.HOURS).plus(
            minutes,
            ChronoUnit.MINUTES,
        )

    val met =
        Metric {
            metricName = customMetricName
            namespace = customMetricNamespace
        }

    val metStat =
```

```
        MetricStat {
            stat = "Maximum"
            period = 1
            metric = met
        }

    val dataQUery =
        MetricDataQuery {
            metricStat = metStat
            id = "foo2"
            returnData = true
        }

    val dq = ArrayList<MetricDataQuery>()
    dq.add(dataQUery)
    val getMetReq =
        GetMetricDataRequest {
            maxDatapoints = 10
            scanBy = ScanBy.TimestampDescending
            startTime =
                aws.smithy.kotlin.runtime.time
                    .Instant(nowDate)
            endTime =
                aws.smithy.kotlin.runtime.time
                    .Instant(date2)
            metricDataQueries = dq
        }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        val response = cwClient.getMetricData(getMetReq)
        response.metricDataResults?.forEach { item ->
            println("The label is ${item.label}")
            println("The status code is ${item.statusCode}")
        }
    }
}

suspend fun describeAlarms() {
    val typeList = ArrayList<AlarmType>()
    typeList.add(AlarmType.MetricAlarm)
    val alarmsRequest =
        DescribeAlarmsRequest {
            alarmTypes = typeList
            maxRecords = 10
```

```
        }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        val response = cwClient.describeAlarms(alarmsRequest)
        response.metricAlarms?.forEach { alarm ->
            println("Alarm name: ${alarm.alarmName}")
            println("Alarm description: ${alarm.alarmDescription}")
        }
    }
}

suspend fun createAlarm(fileName: String): String {
    // Read values from the JSON file.
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode: JsonNode = ObjectMapper().readTree(parser)
    val customMetricNamespace =
 rootNode.findValue("customMetricNamespace").asText()
    val customMetricName = rootNode.findValue("customMetricName").asText()
    val alarmNameVal = rootNode.findValue("exampleAlarmName").asText()
    val emailTopic = rootNode.findValue("emailTopic").asText()
    val accountId = rootNode.findValue("accountId").asText()
    val region2 = rootNode.findValue("region").asText()

    // Create a List for alarm actions.
    val alarmActionObs: MutableList<String> = ArrayList()
    alarmActionObs.add("arn:aws:sns:$region2:$accountId:$emailTopic")
    val alarmRequest =
        PutMetricAlarmRequest {
            alarmActions = alarmActionObs
            alarmDescription = "Example metric alarm"
            alarmName = alarmNameVal
            comparisonOperator = ComparisonOperator.GreaterThanOrEqualToThreshold
            threshold = 100.00
            metricName = customMetricName
            namespace = customMetricNamespace
            evaluationPeriods = 1
            period = 10
            statistic = Statistic.Maximum
            datapointsToAlarm = 1
            treatMissingData = "ignore"
        }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient.putMetricAlarm(alarmRequest)
```

```
        println("$alarmNameVal was successfully created!")
        return alarmNameVal
    }
}

suspend fun addMetricToDashboard(
    fileNameVal: String,
    dashboardNameVal: String,
) {
    val dashboardRequest =
        PutDashboardRequest {
            dashboardName = dashboardNameVal
            dashboardBody = readFileAsString(fileNameVal)
        }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient.putDashboard(dashboardRequest)
        println("$dashboardNameVal was successfully updated.")
    }
}

suspend fun createNewCustomMetric(dataPoint: Double) {
    val dimension =
        Dimension {
            name = "UNIQUE_PAGES"
            value = "URLS"
        }

    // Set an Instant object.
    val time =
 ZonedDateTime.now(ZoneOffset.UTC).format(DateTimeFormatter.ISO_INSTANT)
    val instant = Instant.parse(time)
    val datum =
        MetricDatum {
            metricName = "PAGES_VISITED"
            unit = StandardUnit.None
            value = dataPoint
            timestamp =
                aws.smithy.kotlin.runtime.time
                    .Instant(instant)
            dimensions = listOf(dimension)
        }

    val request =
```

```
        PutMetricDataRequest {
            namespace = "SITE/TRAFFIC"
            metricData = listOf(datum)
        }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient.putMetricData(request)
        println("Added metric values for for metric PAGES_VISITED")
    }
}

suspend fun listDashboards() {
    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient
            .listDashboardsPaginated({})
            .transform { it.dashboardEntries?.forEach { obj -> emit(obj) } }
            .collect { obj ->
                println("Name is ${obj.dashboardName}")
                println("Dashboard ARN is ${obj.dashboardArn}")
            }
    }
}

suspend fun createDashboardWithMetrics(
    dashboardNameVal: String,
    fileNameVal: String,
) {
    val dashboardRequest =
        PutDashboardRequest {
            dashboardName = dashboardNameVal
            dashboardBody = readFileAsString(fileNameVal)
        }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        val response = cwClient.putDashboard(dashboardRequest)
        println("$dashboardNameVal was successfully created.")
        val messages = response.dashboardValidationMessages
        if (messages != null) {
            if (messages.isEmpty()) {
                println("There are no messages in the new Dashboard")
            } else {
                for (message in messages) {
                    println("Message is: ${message.message}")
                }
```

```
            }
        }
    }
}

fun readFileAsString(file: String): String =
 String(Files.readAllBytes(Paths.get(file)))

suspend fun getMetricStatistics(costDateWeek: String?) {
    val start = Instant.parse(costDateWeek)
    val endDate = Instant.now()
    val dimension =
        Dimension {
            name = "Currency"
            value = "USD"
        }

    val dimensionList: MutableList<Dimension> = ArrayList()
    dimensionList.add(dimension)

    val statisticsRequest =
        GetMetricStatisticsRequest {
            metricName = "EstimatedCharges"
            namespace = "AWS/Billing"
            dimensions = dimensionList
            statistics = listOf(Statistic.Maximum)
            startTime =
                aws.smithy.kotlin.runtime.time
                    .Instant(start)
            endTime =
                aws.smithy.kotlin.runtime.time
                    .Instant(endDate)
            period = 86400
        }
    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        val response = cwClient.getMetricStatistics(statisticsRequest)
        val data: List<Datapoint>? = response.datapoints
        if (data != null) {
            if (!data.isEmpty()) {
                for (datapoint in data) {
                    println("Timestamp:  ${datapoint.timestamp} Maximum value:
 ${datapoint.maximum}")
                }
            } else {
```

```
                println("The returned data list is empty")
            }
        }
    }
}

suspend fun getAndDisplayMetricStatistics(
    nameSpaceVal: String,
    metVal: String,
    metricOption: String,
    date: String,
    myDimension: Dimension,
) {
    val start = Instant.parse(date)
    val endDate = Instant.now()
    val statisticsRequest =
        GetMetricStatisticsRequest {
            endTime =
                aws.smithy.kotlin.runtime.time
                    .Instant(endDate)
            startTime =
                aws.smithy.kotlin.runtime.time
                    .Instant(start)
            dimensions = listOf(myDimension)
            metricName = metVal
            namespace = nameSpaceVal
            period = 86400
            statistics = listOf(Statistic.fromValue(metricOption))
        }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        val response = cwClient.getMetricStatistics(statisticsRequest)
        val data = response.datapoints
        if (data != null) {
            if (data.isNotEmpty()) {
                for (datapoint in data) {
                    println("Timestamp: ${datapoint.timestamp} Maximum value:
 ${datapoint.maximum}")
                }
            } else {
                println("The returned data list is empty")
            }
        }
    }
```

```
}

suspend fun listMets(namespaceVal: String?): ArrayList<String>? {
    val metList = ArrayList<String>()
    val request =
        ListMetricsRequest {
            namespace = namespaceVal
        }
    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        val reponse = cwClient.listMetrics(request)
        reponse.metrics?.forEach { metrics ->
            val data = metrics.metricName
            if (!metList.contains(data)) {
                metList.add(data!!)
            }
        }
    }
    return metList
}

suspend fun getSpecificMet(namespaceVal: String?): Dimension? {
    val request =
        ListMetricsRequest {
            namespace = namespaceVal
        }
    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        val response = cwClient.listMetrics(request)
        val myList = response.metrics
        if (myList != null) {
            return myList[0].dimensions?.get(0)
        }
    }
    return null
}

suspend fun listNameSpaces(): ArrayList<String> {
    val nameSpaceList = ArrayList<String>()
    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        val response = cwClient.listMetrics(ListMetricsRequest {})
        response.metrics?.forEach { metrics ->
            val data = metrics.namespace
            if (!nameSpaceList.contains(data)) {
                nameSpaceList.add(data!!)
            }
```

```
        }
    }
    return nameSpaceList
}
```

- For API details, see the following topics in *Amazon SDK for Kotlin API reference*.
  - DeleteAlarms
  - DeleteAnomalyDetector
  - DeleteDashboards
  - DescribeAlarmHistory
  - DescribeAlarms
  - DescribeAlarmsForMetric
  - DescribeAnomalyDetectors
  - GetMetricData
  - GetMetricStatistics
  - GetMetricWidgetImage
  - ListMetrics
  - PutAnomalyDetector
  - PutDashboard
  - PutMetricAlarm
  - PutMetricData

For a complete list of Amazon SDK developer guides and code examples, see Using CloudWatch with an Amazon SDK. This topic also includes information about getting started and details about previous SDK versions.

Actions for CloudWatch using Amazon SDKs

The following code examples demonstrate how to perform individual CloudWatch actions with Amazon SDKs. Each example includes a link to GitHub, where you can find instructions for setting up and running the code.

These excerpts call the CloudWatch API and are code excerpts from larger programs that must be run in context. You can see actions in context in Scenarios for CloudWatch using Amazon SDKs .

The following examples include only the most commonly used actions. For a complete list, see the Amazon CloudWatch API Reference.

Examples

- Use DeleteAlarms with an Amazon SDK or CLI
- Use DeleteAnomalyDetector with an Amazon SDK or CLI
- Use DeleteDashboards with an Amazon SDK or CLI
- Use DescribeAlarmHistory with an Amazon SDK or CLI
- Use DescribeAlarms with an Amazon SDK or CLI
- Use DescribeAlarmsForMetric with an Amazon SDK or CLI
- Use DescribeAnomalyDetectors with an Amazon SDK or CLI
- Use DisableAlarmActions with an Amazon SDK or CLI
- Use EnableAlarmActions with an Amazon SDK or CLI
- Use GetDashboard with an Amazon SDK or CLI
- Use GetMetricData with an Amazon SDK or CLI
- Use GetMetricStatistics with an Amazon SDK or CLI
- Use GetMetricWidgetImage with an Amazon SDK or CLI
- Use ListDashboards with an Amazon SDK or CLI
- Use ListMetrics with an Amazon SDK or CLI
- Use PutAnomalyDetector with an Amazon SDK or CLI
- Use PutDashboard with an Amazon SDK or CLI
- Use PutMetricAlarm with an Amazon SDK or CLI
- Use PutMetricData with an Amazon SDK or CLI

Use `DeleteAlarms` with an Amazon SDK or CLI

The following code examples show how to use `DeleteAlarms`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Learn the basics
- Get started with alarms

- Manage metrics and alarms

.NET

Amazon SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

```
/// <summary>
/// Delete a list of alarms from CloudWatch.
/// </summary>
/// <param name="alarmNames">A list of names of alarms to delete.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DeleteAlarms(List<string> alarmNames)
{
    var deleteAlarmsResult = await _amazonCloudWatch.DeleteAlarmsAsync(
        new DeleteAlarmsRequest()
        {
            AlarmNames = alarmNames
        });

    return deleteAlarmsResult.HttpStatusCode == HttpStatusCode.OK;
}
```

- For API details, see DeleteAlarms in *Amazon SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

Include the required files.

```
#include <aws/core/Aws.h>
#include <aws/monitoring/CloudWatchClient.h>
#include <aws/monitoring/model/DeleteAlarmsRequest.h>
#include <iostream>
```

Delete the alarm.

```
        Aws::CloudWatch::CloudWatchClient cw;
        Aws::CloudWatch::Model::DeleteAlarmsRequest request;
        request.AddAlarmNames(alarm_name);

        auto outcome = cw.DeleteAlarms(request);
        if (!outcome.IsSuccess())
        {
            std::cout << "Failed to delete CloudWatch alarm:" <<
                outcome.GetError().GetMessage() << std::endl;
        }
        else
        {
            std::cout << "Successfully deleted CloudWatch alarm " << alarm_name
                << std::endl;
        }
```

- For API details, see [DeleteAlarms] in *Amazon SDK for C++ API Reference*.

CLI

Amazon CLI

To delete an alarm

The following example uses the `delete-alarms` command to delete the Amazon CloudWatch alarm named "myalarm":

```
aws cloudwatch delete-alarms --alarm-names myalarm
```

Output:

```
This command returns to the prompt if successful.
```

- For API details, see DeleteAlarms in *Amazon CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

```
    /**
     * Deletes a CloudWatch alarm.
     *
     * @param alarmName the name of the alarm to be deleted
     * @return a {@link CompletableFuture} representing the asynchronous
 operation to delete the alarm
     * the {@link DeleteAlarmsResponse} is returned when the operation completes
 successfully,
     * or a {@link RuntimeException} is thrown if the operation fails
     */
    public CompletableFuture<DeleteAlarmsResponse> deleteCWAlarmAsync(String
 alarmName) {
        DeleteAlarmsRequest request = DeleteAlarmsRequest.builder()
            .alarmNames(alarmName)
            .build();

        return getAsyncClient().deleteAlarms(request)
            .whenComplete((response, exception) -> {
                if (exception != null) {
                    throw new RuntimeException("Failed to delete the alarm:{} " +
 alarmName, exception);
                } else {
                    logger.info("Successfully deleted alarm {} ", alarmName);
                }
            });
    }
```

- For API details, see DeleteAlarms in *Amazon SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

Import the SDK and client modules and call the API.

```
import { DeleteAlarmsCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new DeleteAlarmsCommand({
    AlarmNames: [process.env.CLOUDWATCH_ALARM_NAME], // Set the value of
 CLOUDWATCH_ALARM_NAME to the name of an existing alarm.
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

Create the client in a separate module and export it.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- For more information, see Amazon SDK for JavaScript Developer Guide.
- For API details, see DeleteAlarms in *Amazon SDK for JavaScript API Reference*.

SDK for JavaScript (v2)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

Import the SDK and client modules and call the API.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatch service object
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });

var params = {
  AlarmNames: ["Web_Server_CPU_Utilization"],
};

cw.deleteAlarms(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- For more information, see Amazon SDK for JavaScript Developer Guide.
- For API details, see DeleteAlarms in *Amazon SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

```
suspend fun deleteAlarm(alarmNameVal: String) {
    val request =
        DeleteAlarmsRequest {
            alarmNames = listOf(alarmNameVal)
        }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient.deleteAlarms(request)
        println("Successfully deleted alarm $alarmNameVal")
    }
}
```

- For API details, see DeleteAlarms in *Amazon SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

```
class CloudWatchWrapper:
    """Encapsulates Amazon CloudWatch functions."""

    def __init__(self, cloudwatch_resource):
        """
```

```
        :param cloudwatch_resource: A Boto3 CloudWatch resource.
        """
        self.cloudwatch_resource = cloudwatch_resource


    def delete_metric_alarms(self, metric_namespace, metric_name):
        """
        Deletes all of the alarms that are currently watching the specified
 metric.

        :param metric_namespace: The namespace of the metric.
        :param metric_name: The name of the metric.
        """
        try:
            metric = self.cloudwatch_resource.Metric(metric_namespace,
 metric_name)
            metric.alarms.delete()
            logger.info(
                "Deleted alarms for metric %s.%s.", metric_namespace, metric_name
            )
        except ClientError:
            logger.exception(
                "Couldn't delete alarms for metric %s.%s.",
                metric_namespace,
                metric_name,
            )
            raise
```

- For API details, see DeleteAlarms in *Amazon SDK for Python (Boto3) API Reference*.

SAP ABAP

SDK for SAP ABAP

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

```
TRY.
    lo_cwt->deletealarms(
      it_alarmnames = it_alarm_names ).
    MESSAGE 'Alarms deleted.' TYPE 'I'.
  CATCH /aws1/cx_cwtresourcenotfound.
    MESSAGE 'Resource being accessed is not found.' TYPE 'E'.
ENDTRY.
```

- For API details, see DeleteAlarms in *Amazon SDK for SAP ABAP API reference*.

For a complete list of Amazon SDK developer guides and code examples, see Using CloudWatch with an Amazon SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DeleteAnomalyDetector` with an Amazon SDK or CLI

The following code examples show how to use `DeleteAnomalyDetector`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Learn the basics

.NET

Amazon SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

```
/// <summary>
/// Delete a single metric anomaly detector.
/// </summary>
/// <param name="anomalyDetector">The anomaly detector to delete.</param>
/// <returns>True if successful.</returns>
```

```
    public async Task<bool> DeleteAnomalyDetector(SingleMetricAnomalyDetector
 anomalyDetector)
    {
        var deleteAnomalyDetectorResponse = await
 _amazonCloudWatch.DeleteAnomalyDetectorAsync(
            new DeleteAnomalyDetectorRequest()
            {
                SingleMetricAnomalyDetector = anomalyDetector
            });

        return deleteAnomalyDetectorResponse.HttpStatusCode == HttpStatusCode.OK;
    }
```

- For API details, see [DeleteAnomalyDetector](#) in *Amazon SDK for .NET API Reference*.

CLI

Amazon CLI

To delete a specified anomaly detection model

The following `delete-anomaly-detector` example deletes an anomaly detector model in the specified account.

```
aws cloudwatch delete-anomaly-detector \
    --namespace AWS/Logs \
    --metric-name IncomingBytes \
    --stat SampleCount
```

This command produces no output.

For more information, see [Deleting an anomaly detection model](#) in the *Amazon CloudWatch User Guide*.

- For API details, see [DeleteAnomalyDetector](#) in *Amazon CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [Amazon Code Examples Repository](https://github.com/awsdocs/aws-doc-sdk-examples/tree/main/javav2/example_code/cloudwatch#code-examples).

```
    /**
     * Deletes an Anomaly Detector.
     *
     * @param fileName the name of the file containing the Anomaly Detector
configuration
     * @return a CompletableFuture that represents the asynchronous deletion of
the Anomaly Detector
     */
    public CompletableFuture<DeleteAnomalyDetectorResponse>
deleteAnomalyDetectorAsync(String fileName) {
        CompletableFuture<JsonNode> readFileFuture =
CompletableFuture.supplyAsync(() -> {
            try {
                JsonParser parser = new JsonFactory().createParser(new
File(fileName));
                return new ObjectMapper().readTree(parser); // Return the root
node
            } catch (IOException e) {
                throw new RuntimeException("Failed to read or parse the file",
e);
            }
        });

        return readFileFuture.thenCompose(rootNode -> {
            String customMetricNamespace =
rootNode.findValue("customMetricNamespace").asText();
            String customMetricName =
rootNode.findValue("customMetricName").asText();

            SingleMetricAnomalyDetector singleMetricAnomalyDetector =
SingleMetricAnomalyDetector.builder()
                .metricName(customMetricName)
```

```
                .namespace(customMetricNamespace)
                .stat("Maximum")
                .build();

            DeleteAnomalyDetectorRequest request =
 DeleteAnomalyDetectorRequest.builder()
                .singleMetricAnomalyDetector(singleMetricAnomalyDetector)
                .build();

            return getAsyncClient().deleteAnomalyDetector(request);
        }).whenComplete((result, exception) -> {
            if (exception != null) {
                throw new RuntimeException("Failed to delete the Anomaly
 Detector", exception);
            } else {
                logger.info("Successfully deleted the Anomaly Detector.");
            }
        });
    }
```

- For API details, see DeleteAnomalyDetector in *Amazon SDK for Java 2.x API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

```
suspend fun deleteAnomalyDetector(fileName: String) {
    // Read values from the JSON file.
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val customMetricNamespace =
 rootNode.findValue("customMetricNamespace").asText()
    val customMetricName = rootNode.findValue("customMetricName").asText()

    val singleMetricAnomalyDetectorVal =
```

```
        SingleMetricAnomalyDetector {
            metricName = customMetricName
            namespace = customMetricNamespace
            stat = "Maximum"
        }

    val request =
        DeleteAnomalyDetectorRequest {
            singleMetricAnomalyDetector = singleMetricAnomalyDetectorVal
        }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient.deleteAnomalyDetector(request)
        println("Successfully deleted the Anomaly Detector.")
    }
}
```

- For API details, see DeleteAnomalyDetector in *Amazon SDK for Kotlin API reference*.

For a complete list of Amazon SDK developer guides and code examples, see Using CloudWatch with an Amazon SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DeleteDashboards` with an Amazon SDK or CLI

The following code examples show how to use `DeleteDashboards`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Learn the basics

.NET

Amazon SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

```
    /// <summary>
    /// Delete a list of CloudWatch dashboards.
    /// </summary>
    /// <param name="dashboardNames">List of dashboard names to delete.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteDashboards(List<string> dashboardNames)
    {
        var deleteDashboardsResponse = await
 _amazonCloudWatch.DeleteDashboardsAsync(
            new DeleteDashboardsRequest()
            {
                DashboardNames = dashboardNames
            });

        return deleteDashboardsResponse.HttpStatusCode == HttpStatusCode.OK;
    }
```

- For API details, see DeleteDashboards in *Amazon SDK for .NET API Reference*.

CLI

Amazon CLI

To delete specified dashboards

The following `delete-dashboards` example deletes two dashboards named `Dashboard-A` and `Dashboard-B` in the specified account.

```
aws cloudwatch delete-dashboards \
    --dashboard-names Dashboard-A Dashboard-B
```

This command produces no output.

For more information, see Amazon CloudWatch dashboards in the *Amazon CloudWatch User Guide*.

- For API details, see DeleteDashboards in *Amazon CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

```
    /**
     * Deletes the specified dashboard.
     *
     * @param dashboardName the name of the dashboard to be deleted
     * @return a {@link CompletableFuture} representing the asynchronous
 operation of deleting the dashboard
     * @throws RuntimeException if the dashboard deletion fails
     */
    public CompletableFuture<DeleteDashboardsResponse>
 deleteDashboardAsync(String dashboardName) {
        DeleteDashboardsRequest dashboardsRequest =
 DeleteDashboardsRequest.builder()
            .dashboardNames(dashboardName)
            .build();

        return getAsyncClient().deleteDashboards(dashboardsRequest)
            .whenComplete((response, exception) -> {
                if (exception != null) {
                    throw new RuntimeException("Failed to delete the dashboard: "
 + dashboardName, exception);
                } else {
                    logger.info("{} was successfully deleted.", dashboardName);
                }
            });
    }
```

- For API details, see DeleteDashboards in *Amazon SDK for Java 2.x API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

```
suspend fun deleteDashboard(dashboardName: String) {
    val dashboardsRequest =
        DeleteDashboardsRequest {
            dashboardNames = listOf(dashboardName)
        }
    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient.deleteDashboards(dashboardsRequest)
        println("$dashboardName was successfully deleted.")
    }
}
```

- For API details, see DeleteDashboards in *Amazon SDK for Kotlin API reference*.

PowerShell

Tools for PowerShell

Example 1: Deletes the specified dashboard, promoting for confirmation before proceeding. To bypass confirmation add the -Force switch to the command.

```
Remove-CWDashboard -DashboardName Dashboard1
```

- For API details, see DeleteDashboards in *Amazon Tools for PowerShell Cmdlet Reference*.

For a complete list of Amazon SDK developer guides and code examples, see [Using CloudWatch with an Amazon SDK](). This topic also includes information about getting started and details about previous SDK versions.

Use `DescribeAlarmHistory` with an Amazon SDK or CLI

The following code examples show how to use `DescribeAlarmHistory`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- [Learn the basics]()

.NET

Amazon SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [Amazon Code Examples Repository]().

```
    /// <summary>
    /// Describe the history of an alarm for a number of days in the past.
    /// </summary>
    /// <param name="alarmName">The name of the alarm.</param>
    /// <param name="historyDays">The number of days in the past.</param>
    /// <returns>The list of alarm history data.</returns>
    public async Task<List<AlarmHistoryItem>> DescribeAlarmHistory(string
 alarmName, int historyDays)
    {
        List<AlarmHistoryItem> alarmHistory = new List<AlarmHistoryItem>();
        var paginatedAlarmHistory =
 _amazonCloudWatch.Paginators.DescribeAlarmHistory(
            new DescribeAlarmHistoryRequest()
            {
                AlarmName = alarmName,
                EndDateUtc = DateTime.UtcNow,
                HistoryItemType = HistoryItemType.StateUpdate,
                StartDateUtc = DateTime.UtcNow.AddDays(-historyDays)
```

```
            });

        await foreach (var data in paginatedAlarmHistory.AlarmHistoryItems)
        {
            alarmHistory.Add(data);
        }
        return alarmHistory;
    }
```

- For API details, see DescribeAlarmHistory in *Amazon SDK for .NET API Reference*.

CLI

Amazon CLI

To retrieve history for an alarm

The following example uses the `describe-alarm-history` command to retrieve history for the Amazon CloudWatch alarm named "myalarm":

```
aws cloudwatch describe-alarm-history --alarm-name "myalarm" --history-item-type StateUpdate
```

Output:

```
{
    "AlarmHistoryItems": [
        {
            "Timestamp": "2014-04-09T18:59:06.442Z",
            "HistoryItemType": "StateUpdate",
            "AlarmName": "myalarm",
            "HistoryData": "{\"version\":\"1.0\",\"oldState\":{\"stateValue
\":\"ALARM\",\"stateReason\":\"testing purposes\"},\"newState\":{\"stateValue
\":\"OK\",\"stateReason\":\"Threshold Crossed: 2 datapoints were not
 greater than the threshold (70.0). The most recent datapoints: [38.958,
 40.292].\",\"stateReasonData\":{\"version\":\"1.0\",\"queryDate\":
\"2014-04-09T18:59:06.419+0000\",\"startDate\":\"2014-04-09T18:44:00.000+0000\",
\"statistic\":\"Average\",\"period\":300,\"recentDatapoints\":[38.958,40.292],
\"threshold\":70.0}}}",
            "HistorySummary": "Alarm updated from ALARM to OK"
        },
```

```
        {
            "Timestamp": "2014-04-09T18:59:05.805Z",
            "HistoryItemType": "StateUpdate",
            "AlarmName": "myalarm",
            "HistoryData": "{\"version\":\"1.0\",\"oldState\":{\"stateValue
\":\"OK\",\"stateReason\":\"Threshold Crossed: 2 datapoints were
 not greater than the threshold (70.0). The most recent datapoints:
 [38.839999999999996, 39.714].\",\"stateReasonData\":{\"version\":
\"1.0\",\"queryDate\":\"2014-03-11T22:45:41.569+0000\",\"startDate\":
\"2014-03-11T22:30:00.000+0000\",\"statistic\":\"Average\",\"period\":300,
\"recentDatapoints\":[38.839999999999996,39.714],\"threshold\":70.0}},\"newState
\":{\"stateValue\":\"ALARM\",\"stateReason\":\"testing purposes\"}}",
            "HistorySummary": "Alarm updated from OK to ALARM"
        }
    ]
}
```

- For API details, see DescribeAlarmHistory in *Amazon CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

```
    /**
     * Retrieves the alarm history for a given alarm name and date range.
     *
     * @param fileName the path to the JSON file containing the alarm name
     * @param date     the date to start the alarm history search (in the format
 "yyyy-MM-dd'T'HH:mm:ss'Z'")
     * @return a {@code CompletableFuture<Void>} that completes when the alarm
 history has been retrieved and processed
     */
    public CompletableFuture<Void> getAlarmHistoryAsync(String fileName, String
 date) {
```

```
        CompletableFuture<String> readFileFuture =
CompletableFuture.supplyAsync(() -> {
            try {
                JsonParser parser = new JsonFactory().createParser(new
File(fileName));
                com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
                return rootNode.findValue("exampleAlarmName").asText(); // Return
alarmName from the JSON file
            } catch (IOException e) {
                throw new RuntimeException("Failed to read or parse the file",
e);
            }
        });

        // Use the alarm name to describe alarm history with a paginator.
        return readFileFuture.thenCompose(alarmName -> {
            try {
                Instant start = Instant.parse(date);
                Instant endDate = Instant.now();
                DescribeAlarmHistoryRequest historyRequest =
DescribeAlarmHistoryRequest.builder()
                    .startDate(start)
                    .endDate(endDate)
                    .alarmName(alarmName)
                    .historyItemType(HistoryItemType.ACTION)
                    .build();

                // Use the paginator to paginate through alarm history pages.
                DescribeAlarmHistoryPublisher historyPublisher =
getAsyncClient().describeAlarmHistoryPaginator(historyRequest);
                CompletableFuture<Void> future = historyPublisher
                    .subscribe(response ->
response.alarmHistoryItems().forEach(item -> {
                        logger.info("History summary: {}",
item.historySummary());
                        logger.info("Timestamp: {}", item.timestamp());
                    }))
                    .whenComplete((result, exception) -> {
                        if (exception != null) {
                            logger.error("Error occurred while getting alarm
history: " + exception.getMessage(), exception);
                        } else {
```

```
                            logger.info("Successfully retrieved all alarm
history.");
                        }
                    });

                // Return the future to the calling code for further handling
                return future;
            } catch (Exception e) {
                throw new RuntimeException("Failed to process alarm history", e);
            }
        }).whenComplete((result, exception) -> {
            if (exception != null) {
                throw new RuntimeException("Error completing alarm history
processing", exception);
            }
        });
    }
```

- For API details, see DescribeAlarmHistory in *Amazon SDK for Java 2.x API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

```
suspend fun getAlarmHistory(
    fileName: String,
    date: String,
) {
    // Read values from the JSON file.
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val alarmNameVal = rootNode.findValue("exampleAlarmName").asText()
    val start = Instant.parse(date)
    val endDateVal = Instant.now()
```

```
    val historyRequest =
        DescribeAlarmHistoryRequest {
            startDate =
                aws.smithy.kotlin.runtime.time
                    .Instant(start)
            endDate =
                aws.smithy.kotlin.runtime.time
                    .Instant(endDateVal)
            alarmName = alarmNameVal
            historyItemType = HistoryItemType.Action
        }

    CloudWatchClient {
        credentialsProvider = EnvironmentCredentialsProvider()
        region = "us-east-1"
    }.use { cwClient ->
        val response = cwClient.describeAlarmHistory(historyRequest)
        val historyItems = response.alarmHistoryItems
        if (historyItems != null) {
            if (historyItems.isEmpty()) {
                println("No alarm history data found for $alarmNameVal.")
            } else {
                for (item in historyItems) {
                    println("History summary ${item.historySummary}")
                    println("Time stamp: ${item.timestamp}")
                }
            }
        }
    }
}
```

- For API details, see DescribeAlarmHistory in *Amazon SDK for Kotlin API reference*.

For a complete list of Amazon SDK developer guides and code examples, see Using CloudWatch with an Amazon SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DescribeAlarms` with an Amazon SDK or CLI

The following code examples show how to use `DescribeAlarms`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Learn the basics
- Get started with alarms

.NET

Amazon SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

```
    /// <summary>
    /// Describe the current alarms, optionally filtered by state.
    /// </summary>
    /// <param name="stateValue">Optional filter for alarm state.</param>
    /// <returns>The list of alarm data.</returns>
    public async Task<List<MetricAlarm>> DescribeAlarms(StateValue? stateValue =
 null)
    {
        List<MetricAlarm> alarms = new List<MetricAlarm>();
        var paginatedDescribeAlarms =
 _amazonCloudWatch.Paginators.DescribeAlarms(
            new DescribeAlarmsRequest()
            {
                StateValue = stateValue
            });

        await foreach (var data in paginatedDescribeAlarms.MetricAlarms)
        {
            alarms.Add(data);
        }
        return alarms;
    }
```

- For API details, see DescribeAlarms in *Amazon SDK for .NET API Reference*.

CLI

Amazon CLI

To list information about an alarm

The following example uses the `describe-alarms` command to provide information about the alarm named "myalarm":

```
aws cloudwatch describe-alarms --alarm-names "myalarm"
```

Output:

```
{
    "MetricAlarms": [
        {
            "EvaluationPeriods": 2,
            "AlarmArn": "arn:aws:cloudwatch:us-
east-1:123456789012:alarm:myalarm",
            "StateUpdatedTimestamp": "2014-04-09T18:59:06.442Z",
            "AlarmConfigurationUpdatedTimestamp": "2012-12-27T00:49:54.032Z",
            "ComparisonOperator": "GreaterThanThreshold",
            "AlarmActions": [
                "arn:aws:sns:us-east-1:123456789012:myHighCpuAlarm"
            ],
            "Namespace": "AWS/EC2",
            "AlarmDescription": "CPU usage exceeds 70 percent",
            "StateReasonData": "{\"version\":\"1.0\",\"queryDate\":
\"2014-04-09T18:59:06.419+0000\",\"startDate\":\"2014-04-09T18:44:00.000+0000\",
\"statistic\":\"Average\",\"period\":300,\"recentDatapoints\":[38.958,40.292],
\"threshold\":70.0}",
            "Period": 300,
            "StateValue": "OK",
            "Threshold": 70.0,
            "AlarmName": "myalarm",
            "Dimensions": [
                {
                    "Name": "InstanceId",
                    "Value": "i-0c986c72"
                }
            ],
            "Statistic": "Average",
```

```
            "StateReason": "Threshold Crossed: 2 datapoints were not greater than
 the threshold (70.0). The most recent datapoints: [38.958, 40.292].",
            "InsufficientDataActions": [],
            "OKActions": [],
            "ActionsEnabled": true,
            "MetricName": "CPUUtilization"
        }
    ]
}
```

- For API details, see DescribeAlarms in *Amazon CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

```
    /**
     * Describes the CloudWatch alarms of the 'METRIC_ALARM' type.
     *
     * @return a {@link CompletableFuture} that represents the asynchronous
 operation
     * of describing the CloudWatch alarms. The future completes when the
     * operation is finished, either successfully or with an error.
     */
    public CompletableFuture<Void> describeAlarmsAsync() {
        List<AlarmType> typeList = new ArrayList<>();
        typeList.add(AlarmType.METRIC_ALARM);
        DescribeAlarmsRequest alarmsRequest = DescribeAlarmsRequest.builder()
            .alarmTypes(typeList)
            .maxRecords(10)
            .build();

        return getAsyncClient().describeAlarms(alarmsRequest)
            .thenAccept(response -> {
                List<MetricAlarm> alarmList = response.metricAlarms();
```

```
                for (MetricAlarm alarm : alarmList) {
                    logger.info("Alarm name: {}", alarm.alarmName());
                    logger.info("Alarm description: {} ",
 alarm.alarmDescription());
                }
            })
            .whenComplete((response, ex) -> {
                if (ex != null) {
                    logger.info("Failed to describe alarms: {}",
 ex.getMessage());
                } else {
                    logger.info("Successfully described alarms.");
                }
            });
    }
```

- For API details, see DescribeAlarms in *Amazon SDK for Java 2.x API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

```
suspend fun describeAlarms() {
    val typeList = ArrayList<AlarmType>()
    typeList.add(AlarmType.MetricAlarm)
    val alarmsRequest =
        DescribeAlarmsRequest {
            alarmTypes = typeList
            maxRecords = 10
        }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        val response = cwClient.describeAlarms(alarmsRequest)
        response.metricAlarms?.forEach { alarm ->
            println("Alarm name: ${alarm.alarmName}")
```

```
            println("Alarm description: ${alarm.alarmDescription}")
        }
    }
}
```

- For API details, see DescribeAlarms in *Amazon SDK for Kotlin API reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

```
require 'aws-sdk-cloudwatch'

# Lists the names of available Amazon CloudWatch alarms.
#
# @param cloudwatch_client [Aws::CloudWatch::Client]
#   An initialized CloudWatch client.
# @example
#   list_alarms(Aws::CloudWatch::Client.new(region: 'us-east-1'))
def list_alarms(cloudwatch_client)
  response = cloudwatch_client.describe_alarms
  if response.metric_alarms.count.positive?
    response.metric_alarms.each do |alarm|
      puts alarm.alarm_name
    end
  else
    puts 'No alarms found.'
  end
rescue StandardError => e
  puts "Error getting information about alarms: #{e.message}"
end
```

- For API details, see DescribeAlarms in *Amazon SDK for Ruby API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

```
    TRY.
        oo_result = lo_cwt->describealarms(                 " oo_result is
 returned for testing purposes. "
          it_alarmnames = it_alarm_names ).
        MESSAGE 'Alarms retrieved.' TYPE 'I'.
      CATCH /aws1/cx_rt_service_generic INTO DATA(lo_exception).
        DATA(lv_error) = |"{ lo_exception->av_err_code }" - { lo_exception-
>av_err_msg }|.
        MESSAGE lv_error TYPE 'E'.
    ENDTRY.
```

- For API details, see DescribeAlarms in *Amazon SDK for SAP ABAP API reference*.

For a complete list of Amazon SDK developer guides and code examples, see Using CloudWatch with an Amazon SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DescribeAlarmsForMetric` with an Amazon SDK or CLI

The following code examples show how to use `DescribeAlarmsForMetric`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Learn the basics
- Manage metrics and alarms

.NET

Amazon SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [Amazon Code Examples Repository].

```
    /// <summary>
    /// Describe the current alarms for a specific metric.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metricName">The name of the metric.</param>
    /// <returns>The list of alarm data.</returns>
    public async Task<List<MetricAlarm>> DescribeAlarmsForMetric(string
 metricNamespace, string metricName)
    {
        var alarmsResult = await _amazonCloudWatch.DescribeAlarmsForMetricAsync(
            new DescribeAlarmsForMetricRequest()
            {
                Namespace = metricNamespace,
                MetricName = metricName
            });

        return alarmsResult.MetricAlarms;
    }
```

- For API details, see DescribeAlarmsForMetric in *Amazon SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [Amazon Code Examples Repository].

Include the required files.

```
#include <aws/core/Aws.h>
#include <aws/monitoring/CloudWatchClient.h>
#include <aws/monitoring/model/DescribeAlarmsRequest.h>
#include <aws/monitoring/model/DescribeAlarmsResult.h>
#include <iomanip>
#include <iostream>
```

Describe the alarms.

```
        Aws::CloudWatch::CloudWatchClient cw;
        Aws::CloudWatch::Model::DescribeAlarmsRequest request;
        request.SetMaxRecords(1);

        bool done = false;
        bool header = false;
        while (!done)
        {
            auto outcome = cw.DescribeAlarms(request);
            if (!outcome.IsSuccess())
            {
                std::cout << "Failed to describe CloudWatch alarms:" <<
                    outcome.GetError().GetMessage() << std::endl;
                break;
            }

            if (!header)
            {
                std::cout << std::left <<
                    std::setw(32) << "Name" <<
                    std::setw(64) << "Arn" <<
                    std::setw(64) << "Description" <<
                    std::setw(20) << "LastUpdated" <<
                    std::endl;
                header = true;
            }

            const auto &alarms = outcome.GetResult().GetMetricAlarms();
            for (const auto &alarm : alarms)
            {
                std::cout << std::left <<
```

```
                    std::setw(32) << alarm.GetAlarmName() <<
                    std::setw(64) << alarm.GetAlarmArn() <<
                    std::setw(64) << alarm.GetAlarmDescription() <<
                    std::setw(20) <<
                    alarm.GetAlarmConfigurationUpdatedTimestamp().ToGmtString(
                        SIMPLE_DATE_FORMAT_STR) <<
                    std::endl;
            }

            const auto &next_token = outcome.GetResult().GetNextToken();
            request.SetNextToken(next_token);
            done = next_token.empty();
        }
```

- For API details, see DescribeAlarmsForMetric in *Amazon SDK for C++ API Reference*.

CLI

Amazon CLI

To display information about alarms associated with a metric

The following example uses the `describe-alarms-for-metric` command to display information about any alarms associated with the Amazon EC2 CPUUtilization metric and the instance with the ID i-0c986c72.:

```
aws cloudwatch describe-alarms-for-metric --metric-name CPUUtilization --
namespace AWS/EC2 --dimensions Name=InstanceId,Value=i-0c986c72
```

Output:

```
{
    "MetricAlarms": [
        {
            "EvaluationPeriods": 10,
            "AlarmArn": "arn:aws:cloudwatch:us-
east-1:111122223333:alarm:myHighCpuAlarm2",
            "StateUpdatedTimestamp": "2013-10-30T03:03:51.479Z",
            "AlarmConfigurationUpdatedTimestamp": "2013-10-30T03:03:50.865Z",
            "ComparisonOperator": "GreaterThanOrEqualToThreshold",
            "AlarmActions": [
```

```
                "arn:aws:sns:us-east-1:111122223333:NotifyMe"
            ],
            "Namespace": "AWS/EC2",
            "AlarmDescription": "CPU usage exceeds 70 percent",
            "StateReasonData": "{\"version\":\"1.0\",\"queryDate\":
\"2013-10-30T03:03:51.479+0000\",\"startDate\":\"2013-10-30T02:08:00.000+0000\",
\"statistic\":\"Average\",\"period\":300,\"recentDatapoints\":
[40.698,39.612,42.432,39.796,38.816,42.28,42.854,40.088,40.760000000000005,41.316],
\"threshold\":70.0}",
            "Period": 300,
            "StateValue": "OK",
            "Threshold": 70.0,
            "AlarmName": "myHighCpuAlarm2",
            "Dimensions": [
                {
                    "Name": "InstanceId",
                    "Value": "i-0c986c72"
                }
            ],
            "Statistic": "Average",
            "StateReason": "Threshold Crossed: 10 datapoints were not
 greater than or equal to the threshold (70.0). The most recent datapoints:
 [40.760000000000005, 41.316].",
            "InsufficientDataActions": [],
            "OKActions": [],
            "ActionsEnabled": true,
            "MetricName": "CPUUtilization"
        },
        {
            "EvaluationPeriods": 2,
            "AlarmArn": "arn:aws:cloudwatch:us-
east-1:111122223333:alarm:myHighCpuAlarm",
            "StateUpdatedTimestamp": "2014-04-09T18:59:06.442Z",
            "AlarmConfigurationUpdatedTimestamp": "2014-04-09T22:26:05.958Z",
            "ComparisonOperator": "GreaterThanThreshold",
            "AlarmActions": [
                "arn:aws:sns:us-east-1:111122223333:HighCPUAlarm"
            ],
            "Namespace": "AWS/EC2",
            "AlarmDescription": "CPU usage exceeds 70 percent",
            "StateReasonData": "{\"version\":\"1.0\",\"queryDate\":
\"2014-04-09T18:59:06.419+0000\",\"startDate\":\"2014-04-09T18:44:00.000+0000\",
\"statistic\":\"Average\",\"period\":300,\"recentDatapoints\":[38.958,40.292],
\"threshold\":70.0}",
```

```
                "Period": 300,
                "StateValue": "OK",
                "Threshold": 70.0,
                "AlarmName": "myHighCpuAlarm",
                "Dimensions": [
                    {
                        "Name": "InstanceId",
                        "Value": "i-0c986c72"
                    }
                ],
                "Statistic": "Average",
                "StateReason": "Threshold Crossed: 2 datapoints were not greater than
 the threshold (70.0). The most recent datapoints: [38.958, 40.292].",
                "InsufficientDataActions": [],
                "OKActions": [],
                "ActionsEnabled": false,
                "MetricName": "CPUUtilization"
            }
        ]
}
```

- For API details, see [DescribeAlarmsForMetric](#) in *Amazon CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [Amazon Code Examples Repository](#).

```
    /**
     * Checks for a metric alarm in AWS CloudWatch.
     *
     * @param fileName the name of the file containing the JSON configuration for
 the custom metric
     * @return a {@link CompletableFuture} that completes when the check for the
 metric alarm is complete
     */
```

```
    public CompletableFuture<Void> checkForMetricAlarmAsync(String fileName) {
        CompletableFuture<String> readFileFuture =
 CompletableFuture.supplyAsync(() -> {
            try {
                JsonParser parser = new JsonFactory().createParser(new
 File(fileName));
                com.fasterxml.jackson.databind.JsonNode rootNode = new
 ObjectMapper().readTree(parser);
                return rootNode.toString(); // Return JSON as a string for
 further processing
            } catch (IOException e) {
                throw new RuntimeException("Failed to read file", e);
            }
        });

        return readFileFuture.thenCompose(jsonContent -> {
            try {
                com.fasterxml.jackson.databind.JsonNode rootNode = new
 ObjectMapper().readTree(jsonContent);
                String customMetricNamespace =
 rootNode.findValue("customMetricNamespace").asText();
                String customMetricName =
 rootNode.findValue("customMetricName").asText();

                DescribeAlarmsForMetricRequest metricRequest =
 DescribeAlarmsForMetricRequest.builder()
                    .metricName(customMetricName)
                    .namespace(customMetricNamespace)
                    .build();

                return checkForAlarmAsync(metricRequest, customMetricName, 10);

            } catch (IOException e) {
                throw new RuntimeException("Failed to parse JSON content", e);
            }
        }).whenComplete((result, exception) -> {
            if (exception != null) {
                throw new RuntimeException("Error checking metric alarm",
 exception);
            }
        });
    }

    // Recursive method to check for the alarm.
```

```
    /**
     * Checks for the existence of an alarm asynchronously for the specified
metric.
     *
     * @param metricRequest    the request to describe the alarms for the
specified metric
     * @param customMetricName the name of the custom metric to check for an
alarm
     * @param retries          the number of retries to perform if no alarm is
found
     * @return a {@link CompletableFuture} that completes when an alarm is found
or the maximum number of retries has been reached
     */
    private static CompletableFuture<Void>
checkForAlarmAsync(DescribeAlarmsForMetricRequest metricRequest, String
customMetricName, int retries) {
        if (retries == 0) {
            return CompletableFuture.completedFuture(null).thenRun(() ->
                logger.info("No Alarm state found for {} after 10 retries.",
customMetricName)
            );
        }

        return
(getAsyncClient().describeAlarmsForMetric(metricRequest).thenCompose(response ->
{
            if (response.hasMetricAlarms()) {
                logger.info("Alarm state found for {}", customMetricName);
                return CompletableFuture.completedFuture(null); // Alarm found,
complete the future
            } else {
                return CompletableFuture.runAsync(() -> {
                    try {
                        Thread.sleep(20000);
                        logger.info(".");
                    } catch (InterruptedException e) {
                        throw new RuntimeException("Interrupted while waiting to
retry", e);
                    }
                }).thenCompose(v -> checkForAlarmAsync(metricRequest,
customMetricName, retries - 1)); // Recursive call
            }
        }));
```

```
    }
```

- For API details, see [DescribeAlarmsForMetric] in *Amazon SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [Amazon Code Examples Repository].

Import the SDK and client modules and call the API.

```
import { DescribeAlarmsCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new DescribeAlarmsCommand({
    AlarmNames: [process.env.CLOUDWATCH_ALARM_NAME], // Set the value of
 CLOUDWATCH_ALARM_NAME to the name of an existing alarm.
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

Create the client in a separate module and export it.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- For more information, see Amazon SDK for JavaScript Developer Guide.
- For API details, see DescribeAlarmsForMetric in *Amazon SDK for JavaScript API Reference*.

SDK for JavaScript (v2)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatch service object
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });

cw.describeAlarms({ StateValue: "INSUFFICIENT_DATA" }, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    // List the names of all current alarms in the console
    data.MetricAlarms.forEach(function (item, index, array) {
      console.log(item.AlarmName);
    });
  }
});
```

- For more information, see Amazon SDK for JavaScript Developer Guide.
- For API details, see DescribeAlarmsForMetric in *Amazon SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

```
suspend fun checkForMetricAlarm(fileName: String?) {
    // Read values from the JSON file.
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val customMetricNamespace =
 rootNode.findValue("customMetricNamespace").asText()
    val customMetricName = rootNode.findValue("customMetricName").asText()
    var hasAlarm = false
    var retries = 10

    val metricRequest =
        DescribeAlarmsForMetricRequest {
            metricName = customMetricName
            namespace = customMetricNamespace
        }
    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        while (!hasAlarm && retries > 0) {
            val response = cwClient.describeAlarmsForMetric(metricRequest)
            if (response.metricAlarms?.count()!! > 0) {
                hasAlarm = true
            }
            retries--
            delay(20000)
            println(".")
        }
        if (!hasAlarm) {
            println("No Alarm state found for $customMetricName after 10
 retries.")
        } else {
            println("Alarm state found for $customMetricName.")
        }
    }
```

```
}
```

- For API details, see DescribeAlarmsForMetric in *Amazon SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

```
class CloudWatchWrapper:
    """Encapsulates Amazon CloudWatch functions."""

    def __init__(self, cloudwatch_resource):
        """
        :param cloudwatch_resource: A Boto3 CloudWatch resource.
        """
        self.cloudwatch_resource = cloudwatch_resource


    def get_metric_alarms(self, metric_namespace, metric_name):
        """
        Gets the alarms that are currently watching the specified metric.

        :param metric_namespace: The namespace of the metric.
        :param metric_name: The name of the metric.
        :returns: An iterator that yields the alarms.
        """
        metric = self.cloudwatch_resource.Metric(metric_namespace, metric_name)
        alarm_iter = metric.alarms.all()
        logger.info("Got alarms for metric %s.%s.", metric_namespace,
 metric_name)
        return alarm_iter
```

- For API details, see DescribeAlarmsForMetric in *Amazon SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

```
#
# @param cloudwatch_client [Aws::CloudWatch::Client]
#   An initialized CloudWatch client.
# @example
#   describe_metric_alarms(Aws::CloudWatch::Client.new(region: 'us-east-1'))
def describe_metric_alarms(cloudwatch_client)
  response = cloudwatch_client.describe_alarms

  if response.metric_alarms.count.positive?
    response.metric_alarms.each do |alarm|
      puts '-' * 16
      puts "Name:           #{alarm.alarm_name}"
      puts "State value:    #{alarm.state_value}"
      puts "State reason:   #{alarm.state_reason}"
      puts "Metric:         #{alarm.metric_name}"
      puts "Namespace:      #{alarm.namespace}"
      puts "Statistic:      #{alarm.statistic}"
      puts "Period:         #{alarm.period}"
      puts "Unit:           #{alarm.unit}"
      puts "Eval. periods:  #{alarm.evaluation_periods}"
      puts "Threshold:      #{alarm.threshold}"
      puts "Comp. operator: #{alarm.comparison_operator}"

      if alarm.key?(:ok_actions) && alarm.ok_actions.count.positive?
        puts 'OK actions:'
        alarm.ok_actions.each do |a|
          puts "  #{a}"
```

```
        end
      end

      if alarm.key?(:alarm_actions) && alarm.alarm_actions.count.positive?
        puts 'Alarm actions:'
        alarm.alarm_actions.each do |a|
          puts "  #{a}"
        end
      end

      if alarm.key?(:insufficient_data_actions) &&
         alarm.insufficient_data_actions.count.positive?
        puts 'Insufficient data actions:'
        alarm.insufficient_data_actions.each do |a|
          puts "  #{a}"
        end
      end

      puts 'Dimensions:'
      if alarm.key?(:dimensions) && alarm.dimensions.count.positive?
        alarm.dimensions.each do |d|
          puts "  Name: #{d.name}, Value: #{d.value}"
        end
      else
        puts '  None for this alarm.'
      end
    end
  else
    puts 'No alarms found.'
  end
rescue StandardError => e
  puts "Error getting information about alarms: #{e.message}"
end

# Example usage:
def run_me
  region = ''

  # Print usage information and then stop.
  if ARGV[0] == '--help' || ARGV[0] == '-h'
    puts 'Usage:   ruby cw-ruby-example-show-alarms.rb REGION'
    puts 'Example: ruby cw-ruby-example-show-alarms.rb us-east-1'
    exit 1
  # If no values are specified at the command prompt, use these default values.
```

```
  elsif ARGV.count.zero?
    region = 'us-east-1'
  # Otherwise, use the values as specified at the command prompt.
  else
    region = ARGV[0]
  end

  cloudwatch_client = Aws::CloudWatch::Client.new(region: region)
  puts 'Available alarms:'
  describe_metric_alarms(cloudwatch_client)
end

run_me if $PROGRAM_NAME == __FILE__
```

- For API details, see DescribeAlarmsForMetric in *Amazon SDK for Ruby API Reference*.

For a complete list of Amazon SDK developer guides and code examples, see Using CloudWatch with an Amazon SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DescribeAnomalyDetectors` with an Amazon SDK or CLI

The following code examples show how to use `DescribeAnomalyDetectors`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Learn the basics

.NET

Amazon SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

```
    /// <summary>
    /// Describe anomaly detectors for a metric and namespace.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metricName">The metric of the anomaly detectors.</param>
    /// <returns>The list of detectors.</returns>
    public async Task<List<AnomalyDetector>> DescribeAnomalyDetectors(string
 metricNamespace, string metricName)
    {
        List<AnomalyDetector> detectors = new List<AnomalyDetector>();
        var paginatedDescribeAnomalyDetectors =
 _amazonCloudWatch.Paginators.DescribeAnomalyDetectors(
            new DescribeAnomalyDetectorsRequest()
            {
                MetricName = metricName,
                Namespace = metricNamespace
            });

        await foreach (var data in
 paginatedDescribeAnomalyDetectors.AnomalyDetectors)
        {
            detectors.Add(data);
        }

        return detectors;
    }
```

- For API details, see [DescribeAnomalyDetectors] in *Amazon SDK for .NET API Reference*.

CLI

Amazon CLI

To retrieve a list of anomaly detection models

The following `describe-anomaly-detectors` example displays information about anomaly detector models that are associated with the `AWS/Logs` namespace in the specified account.

```
aws cloudwatch describe-anomaly-detectors \
```

```
    --namespace AWS/Logs
```

Output:


```
{
    "AnomalyDetectors": [
        {
            "Namespace": "AWS/Logs",
            "MetricName": "IncomingBytes",
            "Dimensions": [],
            "Stat": "SampleCount",
            "Configuration": {
                "ExcludedTimeRanges": []
            },
            "StateValue": "TRAINED",
            "SingleMetricAnomalyDetector": {
                "AccountId": "123456789012",
                "Namespace": "AWS/Logs",
                "MetricName": "IncomingBytes",
                "Dimensions": [],
                "Stat": "SampleCount"
            }
        },
        {
            "Namespace": "AWS/Logs",
            "MetricName": "IncomingBytes",
            "Dimensions": [
                {
                    "Name": "LogGroupName",
                    "Value": "demo"
                }
            ],
            "Stat": "Average",
            "Configuration": {
                "ExcludedTimeRanges": []
            },
            "StateValue": "PENDING_TRAINING",
            "SingleMetricAnomalyDetector": {
                "AccountId": "123456789012",
                "Namespace": "AWS/Logs",
                "MetricName": "IncomingBytes",
                "Dimensions": [
                    {
```

```
                        "Name": "LogGroupName",
                        "Value": "demo"
                    }
                ],
                "Stat": "Average"
            }
        }
    ]
}
```

For more information, see Using CloudWatch anomaly detection in the *Amazon CloudWatch User Guide*.

- For API details, see DescribeAnomalyDetectors in *Amazon CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

```
    /**
     * Describes the anomaly detectors based on the specified JSON file.
     *
     * @param fileName the name of the JSON file containing the custom metric
 namespace and name
     * @return a {@link CompletableFuture} that completes when the anomaly
 detectors have been described
     * @throws RuntimeException if there is a failure during the operation, such
 as when reading or parsing the JSON file,
     *                          or when describing the anomaly detectors
     */
    public CompletableFuture<Void> describeAnomalyDetectorsAsync(String fileName)
 {
        CompletableFuture<JsonNode> readFileFuture =
 CompletableFuture.supplyAsync(() -> {
            try {
```

```
                JsonParser parser = new JsonFactory().createParser(new
File(fileName));
                return new ObjectMapper().readTree(parser);
            } catch (IOException e) {
                throw new RuntimeException("Failed to read or parse the file",
e);
            }
        });

        return readFileFuture.thenCompose(rootNode -> {
            try {
                String customMetricNamespace =
rootNode.findValue("customMetricNamespace").asText();
                String customMetricName =
rootNode.findValue("customMetricName").asText();

                DescribeAnomalyDetectorsRequest detectorsRequest =
DescribeAnomalyDetectorsRequest.builder()
                    .maxResults(10)
                    .metricName(customMetricName)
                    .namespace(customMetricNamespace)
                    .build();

                return
getAsyncClient().describeAnomalyDetectors(detectorsRequest).thenAccept(response
-> {
                    List<AnomalyDetector> anomalyDetectorList =
response.anomalyDetectors();
                    for (AnomalyDetector detector : anomalyDetectorList) {
                        logger.info("Metric name: {} ",
detector.singleMetricAnomalyDetector().metricName());
                        logger.info("State: {} ", detector.stateValue());
                    }
                });
            } catch (RuntimeException e) {
                throw new RuntimeException("Failed to describe anomaly
detectors", e);
            }
        }).whenComplete((result, exception) -> {
            if (exception != null) {
                throw new RuntimeException("Error describing anomaly detectors",
exception);
            }
        });
```

```
    }
```

- For API details, see DescribeAnomalyDetectors in *Amazon SDK for Java 2.x API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

```
suspend fun describeAnomalyDetectors(fileName: String) {
    // Read values from the JSON file.
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val customMetricNamespace =
 rootNode.findValue("customMetricNamespace").asText()
    val customMetricName = rootNode.findValue("customMetricName").asText()

    val detectorsRequest =
        DescribeAnomalyDetectorsRequest {
            maxResults = 10
            metricName = customMetricName
            namespace = customMetricNamespace
        }
    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        val response = cwClient.describeAnomalyDetectors(detectorsRequest)
        response.anomalyDetectors?.forEach { detector ->
            println("Metric name:
 ${detector.singleMetricAnomalyDetector?.metricName}")
            println("State: ${detector.stateValue}")
        }
    }
}
```

- For API details, see DescribeAnomalyDetectors in *Amazon SDK for Kotlin API reference*.

For a complete list of Amazon SDK developer guides and code examples, see Using CloudWatch with an Amazon SDK. This topic also includes information about getting started and details about previous SDK versions.

Use DisableAlarmActions with an Amazon SDK or CLI

The following code examples show how to use DisableAlarmActions.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Get started with alarms
- Manage metrics and alarms

.NET

Amazon SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

```
    /// <summary>
    /// Disable the actions for a list of alarms from CloudWatch.
    /// </summary>
    /// <param name="alarmNames">A list of names of alarms.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DisableAlarmActions(List<string> alarmNames)
    {
        var disableAlarmActionsResult = await
 _amazonCloudWatch.DisableAlarmActionsAsync(
            new DisableAlarmActionsRequest()
            {
                AlarmNames = alarmNames
            });

        return disableAlarmActionsResult.HttpStatusCode == HttpStatusCode.OK;
    }
```

- For API details, see [DisableAlarmActions](#) in *Amazon SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [Amazon Code Examples Repository](#).

Include the required files.

```
#include <aws/core/Aws.h>
#include <aws/monitoring/CloudWatchClient.h>
#include <aws/monitoring/model/DisableAlarmActionsRequest.h>
#include <iostream>
```

Disable the alarm actions.

```
        Aws::CloudWatch::CloudWatchClient cw;

        Aws::CloudWatch::Model::DisableAlarmActionsRequest
 disableAlarmActionsRequest;
        disableAlarmActionsRequest.AddAlarmNames(alarm_name);

        auto disableAlarmActionsOutcome =
 cw.DisableAlarmActions(disableAlarmActionsRequest);
        if (!disableAlarmActionsOutcome.IsSuccess())
        {
            std::cout << "Failed to disable actions for alarm " << alarm_name <<
                ": " << disableAlarmActionsOutcome.GetError().GetMessage() <<
                std::endl;
        }
        else
        {
            std::cout << "Successfully disabled actions for alarm " <<
```

```
                alarm_name << std::endl;
        }
```

- For API details, see DisableAlarmActions in *Amazon SDK for C++ API Reference*.

CLI

Amazon CLI

To disable actions for an alarm

The following example uses the `disable-alarm-actions` command to disable all actions for the alarm named myalarm.:

```
aws cloudwatch disable-alarm-actions --alarm-names myalarm
```

This command returns to the prompt if successful.

- For API details, see DisableAlarmActions in *Amazon CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.cloudwatch.CloudWatchClient;
import software.amazon.awssdk.services.cloudwatch.model.CloudWatchException;
import
 software.amazon.awssdk.services.cloudwatch.model.DisableAlarmActionsRequest;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
```

```
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DisableAlarmActions {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                  <alarmName>

                Where:
                  alarmName - An alarm name to disable (for example, MyAlarm).
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String alarmName = args[0];
        Region region = Region.US_EAST_1;
        CloudWatchClient cw = CloudWatchClient.builder()
                .region(region)
                .build();

        disableActions(cw, alarmName);
        cw.close();
    }

    public static void disableActions(CloudWatchClient cw, String alarmName) {
        try {
            DisableAlarmActionsRequest request =
 DisableAlarmActionsRequest.builder()
                    .alarmNames(alarmName)
                    .build();

            cw.disableAlarmActions(request);
            System.out.printf("Successfully disabled actions on alarm %s",
 alarmName);

        } catch (CloudWatchException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
```

```
                System.exit(1);
            }
        }
}
```

- For API details, see [DisableAlarmActions](#) in *Amazon SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [Amazon Code Examples Repository](#).

Import the SDK and client modules and call the API.

```
import { DisableAlarmActionsCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new DisableAlarmActionsCommand({
    AlarmNames: process.env.CLOUDWATCH_ALARM_NAME, // Set the value of
 CLOUDWATCH_ALARM_NAME to the name of an existing alarm.
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

Create the client in a separate module and export it.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- For more information, see Amazon SDK for JavaScript Developer Guide.
- For API details, see DisableAlarmActions in *Amazon SDK for JavaScript API Reference*.

SDK for JavaScript (v2)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

Import the SDK and client modules and call the API.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatch service object
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });

cw.disableAlarmActions(
  { AlarmNames: ["Web_Server_CPU_Utilization"] },
  function (err, data) {
    if (err) {
      console.log("Error", err);
    } else {
      console.log("Success", data);
    }
  }
);
```

- For more information, see Amazon SDK for JavaScript Developer Guide.
- For API details, see DisableAlarmActions in *Amazon SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [Amazon Code Examples Repository](https://github.com/awsdocs/aws-doc-sdk-examples).

```
suspend fun disableActions(alarmName: String) {
    val request =
        DisableAlarmActionsRequest {
            alarmNames = listOf(alarmName)
        }
    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient.disableAlarmActions(request)
        println("Successfully disabled actions on alarm $alarmName")
    }
}
```

- For API details, see DisableAlarmActions in *Amazon SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

```
class CloudWatchWrapper:
    """Encapsulates Amazon CloudWatch functions."""

    def __init__(self, cloudwatch_resource):
        """
        :param cloudwatch_resource: A Boto3 CloudWatch resource.
```

```
        """
        self.cloudwatch_resource = cloudwatch_resource


    def enable_alarm_actions(self, alarm_name, enable):
        """
        Enables or disables actions on the specified alarm. Alarm actions can be
        used to send notifications or automate responses when an alarm enters a
        particular state.

        :param alarm_name: The name of the alarm.
        :param enable: When True, actions are enabled for the alarm. Otherwise,
 they
                       disabled.
        """
        try:
            alarm = self.cloudwatch_resource.Alarm(alarm_name)
            if enable:
                alarm.enable_actions()
            else:
                alarm.disable_actions()
            logger.info(
                "%s actions for alarm %s.",
                "Enabled" if enable else "Disabled",
                alarm_name,
            )
        except ClientError:
            logger.exception(
                "Couldn't %s actions alarm %s.",
                "enable" if enable else "disable",
                alarm_name,
            )
            raise
```

- For API details, see DisableAlarmActions in *Amazon SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

```
# Disables an alarm in Amazon CloudWatch.
#
# Prerequisites.
#
# - The alarm to disable.
#
# @param cloudwatch_client [Aws::CloudWatch::Client]
#   An initialized CloudWatch client.
# @param alarm_name [String] The name of the alarm to disable.
# @return [Boolean] true if the alarm was disabled; otherwise, false.
# @example
#   exit 1 unless alarm_actions_disabled?(
#     Aws::CloudWatch::Client.new(region: 'us-east-1'),
#     'ObjectsInBucket'
#   )
def alarm_actions_disabled?(cloudwatch_client, alarm_name)
  cloudwatch_client.disable_alarm_actions(alarm_names: [alarm_name])
  true
rescue StandardError => e
  puts "Error disabling alarm actions: #{e.message}"
  false
end

# Example usage:
def run_me
  alarm_name = 'ObjectsInBucket'
  alarm_description = 'Objects exist in this bucket for more than 1 day.'
  metric_name = 'NumberOfObjects'
  # Notify this Amazon Simple Notification Service (Amazon SNS) topic when
  # the alarm transitions to the ALARM state.
  alarm_actions = ['arn:aws:sns:us-
east-1:111111111111:Default_CloudWatch_Alarms_Topic']
```

```
namespace = 'AWS/S3'
statistic = 'Average'
dimensions = [
  {
    name: "BucketName",
    value: "amzn-s3-demo-bucket"
  },
  {
    name: 'StorageType',
    value: 'AllStorageTypes'
  }
]
period = 86_400 # Daily (24 hours * 60 minutes * 60 seconds = 86400 seconds).
unit = 'Count'
evaluation_periods = 1 # More than one day.
threshold = 1 # One object.
comparison_operator = 'GreaterThanThreshold' # More than one object.
# Replace us-west-2 with the AWS Region you're using for Amazon CloudWatch.
region = 'us-east-1'

cloudwatch_client = Aws::CloudWatch::Client.new(region: region)

if alarm_created_or_updated?(
  cloudwatch_client,
  alarm_name,
  alarm_description,
  metric_name,
  alarm_actions,
  namespace,
  statistic,
  dimensions,
  period,
  unit,
  evaluation_periods,
  threshold,
  comparison_operator
)
  puts "Alarm '#{alarm_name}' created or updated."
else
  puts "Could not create or update alarm '#{alarm_name}'."
end

if alarm_actions_disabled?(cloudwatch_client, alarm_name)
  puts "Alarm '#{alarm_name}' disabled."
```

```
    else
      puts "Could not disable alarm '#{alarm_name}'."
    end
  end

  run_me if $PROGRAM_NAME == __FILE__
```

- For API details, see DisableAlarmActions in *Amazon SDK for Ruby API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

```
    "Disables actions on the specified alarm. "
    TRY.
        lo_cwt->disablealarmactions(
          it_alarmnames = it_alarm_names ).
        MESSAGE 'Alarm actions disabled.' TYPE 'I'.
      CATCH /aws1/cx_rt_service_generic INTO DATA(lo_exception).
        DATA(lv_error) = |"{ lo_exception->av_err_code }" - { lo_exception-
>av_err_msg }|.
        MESSAGE lv_error TYPE 'E'.
    ENDTRY.
```

- For API details, see DisableAlarmActions in *Amazon SDK for SAP ABAP API reference*.

For a complete list of Amazon SDK developer guides and code examples, see Using CloudWatch with an Amazon SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `EnableAlarmActions` with an Amazon SDK or CLI

The following code examples show how to use `EnableAlarmActions`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Manage metrics and alarms

.NET

Amazon SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

```
    /// <summary>
    /// Enable the actions for a list of alarms from CloudWatch.
    /// </summary>
    /// <param name="alarmNames">A list of names of alarms.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> EnableAlarmActions(List<string> alarmNames)
    {
        var enableAlarmActionsResult = await
 _amazonCloudWatch.EnableAlarmActionsAsync(
            new EnableAlarmActionsRequest()
            {
                AlarmNames = alarmNames
            });

        return enableAlarmActionsResult.HttpStatusCode == HttpStatusCode.OK;
    }
```

- For API details, see EnableAlarmActions in *Amazon SDK for .NET API Reference*.

C++

SDK for C++

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [Amazon Code Examples Repository](#).

Include the required files.

```
#include <aws/core/Aws.h>
#include <aws/monitoring/CloudWatchClient.h>
#include <aws/monitoring/model/EnableAlarmActionsRequest.h>
#include <aws/monitoring/model/PutMetricAlarmRequest.h>
#include <iostream>
```

Enable the alarm actions.

```
    Aws::CloudWatch::CloudWatchClient cw;
    Aws::CloudWatch::Model::PutMetricAlarmRequest request;
    request.SetAlarmName(alarm_name);
    request.SetComparisonOperator(
        Aws::CloudWatch::Model::ComparisonOperator::GreaterThanThreshold);
    request.SetEvaluationPeriods(1);
    request.SetMetricName("CPUUtilization");
    request.SetNamespace("AWS/EC2");
    request.SetPeriod(60);
    request.SetStatistic(Aws::CloudWatch::Model::Statistic::Average);
    request.SetThreshold(70.0);
    request.SetActionsEnabled(false);
    request.SetAlarmDescription("Alarm when server CPU exceeds 70%");
    request.SetUnit(Aws::CloudWatch::Model::StandardUnit::Seconds);
    request.AddAlarmActions(actionArn);

    Aws::CloudWatch::Model::Dimension dimension;
    dimension.SetName("InstanceId");
    dimension.SetValue(instanceId);
    request.AddDimensions(dimension);
```

```
    auto outcome = cw.PutMetricAlarm(request);
    if (!outcome.IsSuccess())
    {
        std::cout << "Failed to create CloudWatch alarm:" <<
            outcome.GetError().GetMessage() << std::endl;
        return;
    }

    Aws::CloudWatch::Model::EnableAlarmActionsRequest enable_request;
    enable_request.AddAlarmNames(alarm_name);

    auto enable_outcome = cw.EnableAlarmActions(enable_request);
    if (!enable_outcome.IsSuccess())
    {
        std::cout << "Failed to enable alarm actions:" <<
            enable_outcome.GetError().GetMessage() << std::endl;
        return;
    }

    std::cout << "Successfully created alarm " << alarm_name <<
        " and enabled actions on it." << std::endl;
```

- For API details, see EnableAlarmActions in *Amazon SDK for C++ API Reference*.

CLI

Amazon CLI

To enable all actions for an alarm

The following example uses the `enable-alarm-actions` command to enable all actions for the alarm named myalarm.:

```
aws cloudwatch enable-alarm-actions --alarm-names myalarm
```

This command returns to the prompt if successful.

- For API details, see EnableAlarmActions in *Amazon CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [Amazon Code Examples Repository](#).

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.cloudwatch.CloudWatchClient;
import software.amazon.awssdk.services.cloudwatch.model.CloudWatchException;
import
 software.amazon.awssdk.services.cloudwatch.model.EnableAlarmActionsRequest;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class EnableAlarmActions {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                  <alarmName>

                Where:
                  alarmName - An alarm name to enable (for example, MyAlarm).
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String alarm = args[0];
```

```
        Region region = Region.US_EAST_1;
        CloudWatchClient cw = CloudWatchClient.builder()
                .region(region)
                .build();

        enableActions(cw, alarm);
        cw.close();
    }

    public static void enableActions(CloudWatchClient cw, String alarm) {
        try {
            EnableAlarmActionsRequest request =
 EnableAlarmActionsRequest.builder()
                    .alarmNames(alarm)
                    .build();

            cw.enableAlarmActions(request);
            System.out.printf("Successfully enabled actions on alarm %s", alarm);

        } catch (CloudWatchException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- For API details, see EnableAlarmActions in *Amazon SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

Import the SDK and client modules and call the API.

```
import { EnableAlarmActionsCommand } from "@aws-sdk/client-cloudwatch";
```

```
import { client } from "../libs/client.js";

const run = async () => {
  const command = new EnableAlarmActionsCommand({
    AlarmNames: [process.env.CLOUDWATCH_ALARM_NAME], // Set the value of
 CLOUDWATCH_ALARM_NAME to the name of an existing alarm.
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

Create the client in a separate module and export it.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- For more information, see Amazon SDK for JavaScript Developer Guide.
- For API details, see EnableAlarmActions in *Amazon SDK for JavaScript API Reference*.

SDK for JavaScript (v2)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

Import the SDK and client modules and call the API.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });
```

```
// Create CloudWatch service object
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });

var params = {
  AlarmName: "Web_Server_CPU_Utilization",
  ComparisonOperator: "GreaterThanThreshold",
  EvaluationPeriods: 1,
  MetricName: "CPUUtilization",
  Namespace: "AWS/EC2",
  Period: 60,
  Statistic: "Average",
  Threshold: 70.0,
  ActionsEnabled: true,
  AlarmActions: ["ACTION_ARN"],
  AlarmDescription: "Alarm when server CPU exceeds 70%",
  Dimensions: [
    {
      Name: "InstanceId",
      Value: "INSTANCE_ID",
    },
  ],
  Unit: "Percent",
};

cw.putMetricAlarm(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Alarm action added", data);
    var paramsEnableAlarmAction = {
      AlarmNames: [params.AlarmName],
    };
    cw.enableAlarmActions(paramsEnableAlarmAction, function (err, data) {
      if (err) {
        console.log("Error", err);
      } else {
        console.log("Alarm action enabled", data);
      }
    });
  }
});
```

- For more information, see Amazon SDK for JavaScript Developer Guide.
- For API details, see EnableAlarmActions in *Amazon SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

```
suspend fun enableActions(alarm: String) {
    val request =
        EnableAlarmActionsRequest {
            alarmNames = listOf(alarm)
        }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient.enableAlarmActions(request)
        println("Successfully enabled actions on alarm $alarm")
    }
}
```

- For API details, see EnableAlarmActions in *Amazon SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

```
class CloudWatchWrapper:
```

```
    """Encapsulates Amazon CloudWatch functions."""

    def __init__(self, cloudwatch_resource):
        """
        :param cloudwatch_resource: A Boto3 CloudWatch resource.
        """
        self.cloudwatch_resource = cloudwatch_resource


    def enable_alarm_actions(self, alarm_name, enable):
        """
        Enables or disables actions on the specified alarm. Alarm actions can be
        used to send notifications or automate responses when an alarm enters a
        particular state.

        :param alarm_name: The name of the alarm.
        :param enable: When True, actions are enabled for the alarm. Otherwise,
  they
                       disabled.
        """
        try:
            alarm = self.cloudwatch_resource.Alarm(alarm_name)
            if enable:
                alarm.enable_actions()
            else:
                alarm.disable_actions()
            logger.info(
                "%s actions for alarm %s.",
                "Enabled" if enable else "Disabled",
                alarm_name,
            )
        except ClientError:
            logger.exception(
                "Couldn't %s actions alarm %s.",
                "enable" if enable else "disable",
                alarm_name,
            )
            raise
```

- For API details, see EnableAlarmActions in *Amazon SDK for Python (Boto3) API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [Amazon Code Examples Repository](#).

```
    "Enable actions on the specified alarm."
    TRY.
        lo_cwt->enablealarmactions(
          it_alarmnames = it_alarm_names ).
        MESSAGE 'Alarm actions enabled.' TYPE 'I'.
      CATCH /aws1/cx_rt_service_generic INTO DATA(lo_exception).
        DATA(lv_error) = |"{ lo_exception->av_err_code }" - { lo_exception-
>av_err_msg }|.
        MESSAGE lv_error TYPE 'E'.
    ENDTRY.
```

- For API details, see EnableAlarmActions in *Amazon SDK for SAP ABAP API reference*.

For a complete list of Amazon SDK developer guides and code examples, see Using CloudWatch with an Amazon SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `GetDashboard` with an Amazon SDK or CLI

The following code examples show how to use `GetDashboard`.

.NET

Amazon SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [Amazon Code Examples Repository](https://github.com/awsdocs/aws-doc-sdk-examples/tree/main/dotnetv3/CloudWatch#code-examples).

```
/// <summary>
/// Get information on a dashboard.
/// </summary>
/// <param name="dashboardName">The name of the dashboard.</param>
/// <returns>A JSON object with dashboard information.</returns>
public async Task<string> GetDashboard(string dashboardName)
{
    var dashboardResponse = await _amazonCloudWatch.GetDashboardAsync(
        new GetDashboardRequest()
        {
            DashboardName = dashboardName
        });

    return dashboardResponse.DashboardBody;
}
```

- For API details, see [GetDashboard](https://docs.aws.amazon.com/goto/DotNetSDKV3/monitoring-2010-08-01/GetDashboard) in *Amazon SDK for .NET API Reference*.

CLI

Amazon CLI

To retrieve information about a Dashboard

The following `get-dashboard` example displays information about the dashboard named `Dashboard-A` in the specified account.

```
aws cloudwatch get-dashboard \
    --dashboard-name Dashboard-A
```

Output:

```
{
    "DashboardArn": "arn:aws:cloudwatch::123456789012:dashboard/Dashboard-A",
    "DashboardBody": "{\"widgets\":[{\"type\":\"metric\",\"x\":0,\"y\":0,\"width
\":6,\"height\":6,\"properties\":{\"view\":\"timeSeries\",\"stacked\":false,
\"metrics\":[[\"AWS/EC2\",\"NetworkIn\",\"InstanceId\",\"i-0131f062232ade043\"],
[\".\",\"NetworkOut\",\".\",\".\"]],\"region\":\"us-east-1\"}}]}",
    "DashboardName": "Dashboard-A"
}
```

For more information, see Amazon CloudWatch dashboards in the *Amazon CloudWatch User Guide*.

- For API details, see GetDashboard in *Amazon CLI Command Reference*.

PowerShell

Tools for PowerShell

Example 1: Returns the arn the body of the specified dashboard.

```
Get-CWDashboard -DashboardName Dashboard1
```

Output:

```
DashboardArn                                          DashboardBody
------------                                          -------------
arn:aws:cloudwatch::123456789012:dashboard/Dashboard1 {...
```

- For API details, see GetDashboard in *Amazon Tools for PowerShell Cmdlet Reference*.

For a complete list of Amazon SDK developer guides and code examples, see Using CloudWatch with an Amazon SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `GetMetricData` with an Amazon SDK or CLI

The following code examples show how to use `GetMetricData`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Learn the basics

.NET

Amazon SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

```
    /// <summary>
    /// Get data for CloudWatch metrics.
    /// </summary>
    /// <param name="minutesOfData">The number of minutes of data to include.</
param>
    /// <param name="useDescendingTime">True to return the data descending by
 time.</param>
    /// <param name="endDateUtc">The end date for the data, in UTC.</param>
    /// <param name="maxDataPoints">The maximum data points to include.</param>
    /// <param name="dataQueries">Optional data queries to include.</param>
    /// <returns>A list of the requested metric data.</returns>
    public async Task<List<MetricDataResult>> GetMetricData(int minutesOfData,
 bool useDescendingTime, DateTime? endDateUtc = null,
        int maxDataPoints = 0, List<MetricDataQuery>? dataQueries = null)
    {
        var metricData = new List<MetricDataResult>();
        // If no end time is provided, use the current time for the end time.
        endDateUtc ??= DateTime.UtcNow;
        var timeZoneOffset =
 TimeZoneInfo.Local.GetUtcOffset(endDateUtc.Value.ToLocalTime());
        var startTimeUtc = endDateUtc.Value.AddMinutes(-minutesOfData);
        // The timezone string should be in the format +0000, so use the timezone
 offset to format it correctly.
        var timeZoneString = $"{timeZoneOffset.Hours:D2}
{timeZoneOffset.Minutes:D2}";
        var paginatedMetricData = _amazonCloudWatch.Paginators.GetMetricData(
```

```
            new GetMetricDataRequest()
            {
                StartTimeUtc = startTimeUtc,
                EndTimeUtc = endDateUtc.Value,
                LabelOptions = new LabelOptions { Timezone = timeZoneString },
                ScanBy = useDescendingTime ? ScanBy.TimestampDescending :
 ScanBy.TimestampAscending,
                MaxDatapoints = maxDataPoints,
                MetricDataQueries = dataQueries,
            });

        await foreach (var data in paginatedMetricData.MetricDataResults)
        {
            metricData.Add(data);
        }
        return metricData;
    }
```

- For API details, see GetMetricData in *Amazon SDK for .NET API Reference*.

CLI

Amazon CLI

Example 1: To get the Average Total IOPS for the specified EC2 using math expression

The following `get-metric-data` example retrieves CloudWatch metric values for the EC2 instance with InstanceID `i-abcdef` using metric math exprssion that combines `EBSReadOps` and `EBSWriteOps` metrics.

```
aws cloudwatch get-metric-data \
    --metric-data-queries file://file.json \
    --start-time 2024-09-29T22:10:00Z \
    --end-time 2024-09-29T22:15:00Z
```

Contents of `file.json`:

```
[
    {
        "Id": "m3",
        "Expression": "(m1+m2)/300",
```

```
            "Label": "Avg Total IOPS"
        },
        {
            "Id": "m1",
            "MetricStat": {
                "Metric": {
                    "Namespace": "AWS/EC2",
                    "MetricName": "EBSReadOps",
                    "Dimensions": [
                        {
                            "Name": "InstanceId",
                            "Value": "i-abcdef"
                        }
                    ]
                },
                "Period": 300,
                "Stat": "Sum",
                "Unit": "Count"
            },
            "ReturnData": false
        },
        {
            "Id": "m2",
            "MetricStat": {
                "Metric": {
                    "Namespace": "AWS/EC2",
                    "MetricName": "EBSWriteOps",
                    "Dimensions": [
                        {
                            "Name": "InstanceId",
                            "Value": "i-abcdef"
                        }
                    ]
                },
                "Period": 300,
                "Stat": "Sum",
                "Unit": "Count"
            },
            "ReturnData": false
        }
    ]
```

Output:

```
{
    "MetricDataResults": [
        {
            "Id": "m3",
            "Label": "Avg Total IOPS",
            "Timestamps": [
                "2024-09-29T22:10:00+00:00"
            ],
            "Values": [
                96.85
            ],
            "StatusCode": "Complete"
        }
    ],
    "Messages": []
}
```

Example 2: To monitor the estimated Amazon charges using CloudWatch billing metrics

The following `get-metric-data` example retrieves `EstimatedCharges` CloudWatch metric from Amazon/Billing namespace.

```
aws cloudwatch get-metric-data \
    --metric-data-queries '[{"Id":"m1","MetricStat":{"Metric":
{"Namespace":"AWS/Billing","MetricName":"EstimatedCharges","Dimensions":
[{"Name":"Currency","Value":"USD"}]},"Period":21600,"Stat":"Maximum"}}]' \
    --start-time 2024-09-26T12:00:00Z \
    --end-time 2024-09-26T18:00:00Z \
    --region us-east-1
```

Output:

```
{
    "MetricDataResults": [
        {
            "Id": "m1",
            "Label": "EstimatedCharges",
            "Timestamps": [
                "2024-09-26T12:00:00+00:00"
            ],
            "Values": [
                542.38
```

```
            ],
            "StatusCode": "Complete"
        }
    ],
    "Messages": []
}
```


For more information, see Using math expressions with CloudWatch metrics in the *Amazon CloudWatch User Guide*.

- For API details, see GetMetricData in *Amazon CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

```
    /**
     * Retrieves custom metric data from the AWS CloudWatch service.
     *
     * @param fileName the name of the file containing the custom metric
 information
     * @return a {@link CompletableFuture} that completes when the metric data
 has been retrieved
     */
    public CompletableFuture<Void> getCustomMetricDataAsync(String fileName) {
        CompletableFuture<String> readFileFuture =
 CompletableFuture.supplyAsync(() -> {
            try {
                // Read values from the JSON file.
                JsonParser parser = new JsonFactory().createParser(new
 File(fileName));
                com.fasterxml.jackson.databind.JsonNode rootNode = new
 ObjectMapper().readTree(parser);
                return rootNode.toString(); // Return JSON as a string for
 further processing
```

```
            } catch (IOException e) {
                throw new RuntimeException("Failed to read file", e);
            }
        });

        return readFileFuture.thenCompose(jsonContent -> {
            try {
                // Parse the JSON string to extract relevant values.
                com.fasterxml.jackson.databind.JsonNode rootNode = new
 ObjectMapper().readTree(jsonContent);
                String customMetricNamespace =
 rootNode.findValue("customMetricNamespace").asText();
                String customMetricName =
 rootNode.findValue("customMetricName").asText();

                // Set the current time and date range for metric query.
                Instant nowDate = Instant.now();
                long hours = 1;
                long minutes = 30;
                Instant endTime = nowDate.plus(hours,
 ChronoUnit.HOURS).plus(minutes, ChronoUnit.MINUTES);

                Metric met = Metric.builder()
                    .metricName(customMetricName)
                    .namespace(customMetricNamespace)
                    .build();

                MetricStat metStat = MetricStat.builder()
                    .stat("Maximum")
                    .period(60)  // Assuming period in seconds
                    .metric(met)
                    .build();

                MetricDataQuery dataQuery = MetricDataQuery.builder()
                    .metricStat(metStat)
                    .id("foo2")
                    .returnData(true)
                    .build();

                List<MetricDataQuery> dq = new ArrayList<>();
                dq.add(dataQuery);

                GetMetricDataRequest getMetricDataRequest =
 GetMetricDataRequest.builder()
```

```
                    .maxDatapoints(10)
                    .scanBy(ScanBy.TIMESTAMP_DESCENDING)
                    .startTime(nowDate)
                    .endTime(endTime)
                    .metricDataQueries(dq)
                    .build();

                // Call the async method for CloudWatch data retrieval.
                return getAsyncClient().getMetricData(getMetricDataRequest);

            } catch (IOException e) {
                throw new RuntimeException("Failed to parse JSON content", e);
            }
        }).thenAccept(response -> {
            List<MetricDataResult> data = response.metricDataResults();
            for (MetricDataResult item : data) {
                logger.info("The label is: {}", item.label());
                logger.info("The status code is: {}",
 item.statusCode().toString());
            }
        }).exceptionally(exception -> {
            throw new RuntimeException("Failed to get metric data", exception);
        });
    }
```

- For API details, see GetMetricData in *Amazon SDK for Java 2.x API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

```
suspend fun getCustomMetricData(fileName: String) {
    // Read values from the JSON file.
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
```

```
    val customMetricNamespace =
 rootNode.findValue("customMetricNamespace").asText()
    val customMetricName = rootNode.findValue("customMetricName").asText()

    // Set the date.
    val nowDate = Instant.now()
    val hours: Long = 1
    val minutes: Long = 30
    val date2 =
        nowDate.plus(hours, ChronoUnit.HOURS).plus(
            minutes,
            ChronoUnit.MINUTES,
        )

    val met =
        Metric {
            metricName = customMetricName
            namespace = customMetricNamespace
        }

    val metStat =
        MetricStat {
            stat = "Maximum"
            period = 1
            metric = met
        }

    val dataQUery =
        MetricDataQuery {
            metricStat = metStat
            id = "foo2"
            returnData = true
        }

    val dq = ArrayList<MetricDataQuery>()
    dq.add(dataQUery)
    val getMetReq =
        GetMetricDataRequest {
            maxDatapoints = 10
            scanBy = ScanBy.TimestampDescending
            startTime =
                aws.smithy.kotlin.runtime.time
                    .Instant(nowDate)
            endTime =
```

```
                aws.smithy.kotlin.runtime.time
                    .Instant(date2)
            metricDataQueries = dq
        }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        val response = cwClient.getMetricData(getMetReq)
        response.metricDataResults?.forEach { item ->
            println("The label is ${item.label}")
            println("The status code is ${item.statusCode}")
        }
    }
}
```

- For API details, see GetMetricData in *Amazon SDK for Kotlin API reference*.

For a complete list of Amazon SDK developer guides and code examples, see Using CloudWatch with an Amazon SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `GetMetricStatistics` with an Amazon SDK or CLI

The following code examples show how to use `GetMetricStatistics`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Learn the basics
- Manage metrics and alarms

.NET

Amazon SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

```
    /// <summary>
    /// Get billing statistics using a call to a wrapper class.
    /// </summary>
    /// <returns>A collection of billing statistics.</returns>
    private static async Task<List<Datapoint>> SetupBillingStatistics()
    {
        // Make a request for EstimatedCharges with a period of one day for the
 past seven days.
        var billingStatistics = await _cloudWatchWrapper.GetMetricStatistics(
            "AWS/Billing",
            "EstimatedCharges",
            new List<string>() { "Maximum" },
            new List<Dimension>() { new Dimension { Name = "Currency", Value =
 "USD" } },
            7,
            86400);

        billingStatistics = billingStatistics.OrderBy(n => n.Timestamp).ToList();

        return billingStatistics;
    }

    /// <summary>
    /// Wrapper to get statistics for a specific CloudWatch metric.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metricName">The name of the metric.</param>
    /// <param name="statistics">The list of statistics to include.</param>
    /// <param name="dimensions">The list of dimensions to include.</param>
    /// <param name="days">The number of days in the past to include.</param>
    /// <param name="period">The period for the data.</param>
    /// <returns>A list of DataPoint objects for the statistics.</returns>
    public async Task<List<Datapoint>> GetMetricStatistics(string
 metricNamespace,
        string metricName, List<string> statistics, List<Dimension> dimensions,
 int days, int period)
    {
        var metricStatistics = await _amazonCloudWatch.GetMetricStatisticsAsync(
            new GetMetricStatisticsRequest()
            {
                Namespace = metricNamespace,
                MetricName = metricName,
                Dimensions = dimensions,
```

```
                Statistics = statistics,
                StartTimeUtc = DateTime.UtcNow.AddDays(-days),
                EndTimeUtc = DateTime.UtcNow,
                Period = period
            });

        return metricStatistics.Datapoints;
    }
```

- For API details, see GetMetricStatistics in *Amazon SDK for .NET API Reference*.

CLI

Amazon CLI

To get the CPU utilization per EC2 instance

The following example uses the `get-metric-statistics` command to get the CPU utilization for an EC2 instance with the ID i-abcdef.

```
aws cloudwatch get-metric-statistics --metric-name CPUUtilization --start-time 2014-04-08T23:18:00Z --end-time 2014-04-09T23:18:00Z --period 3600 --namespace AWS/EC2 --statistics Maximum --dimensions Name=InstanceId,Value=i-abcdef
```

Output:

```
{
    "Datapoints": [
        {
            "Timestamp": "2014-04-09T11:18:00Z",
            "Maximum": 44.79,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T20:18:00Z",
            "Maximum": 47.92,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T19:18:00Z",
```

```
            "Maximum": 50.85,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T09:18:00Z",
            "Maximum": 47.92,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T03:18:00Z",
            "Maximum": 76.84,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T21:18:00Z",
            "Maximum": 48.96,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T14:18:00Z",
            "Maximum": 47.92,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T08:18:00Z",
            "Maximum": 47.92,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T16:18:00Z",
            "Maximum": 45.55,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T06:18:00Z",
            "Maximum": 47.92,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T13:18:00Z",
            "Maximum": 45.08,
            "Unit": "Percent"
        },
        {
```

```
            "Timestamp": "2014-04-09T05:18:00Z",
            "Maximum": 47.92,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T18:18:00Z",
            "Maximum": 46.88,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T17:18:00Z",
            "Maximum": 52.08,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T07:18:00Z",
            "Maximum": 47.92,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T02:18:00Z",
            "Maximum": 51.23,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T12:18:00Z",
            "Maximum": 47.67,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-08T23:18:00Z",
            "Maximum": 46.88,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T10:18:00Z",
            "Maximum": 51.91,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T04:18:00Z",
            "Maximum": 47.13,
            "Unit": "Percent"
        },
```

```
        {
            "Timestamp": "2014-04-09T15:18:00Z",
            "Maximum": 48.96,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T00:18:00Z",
            "Maximum": 48.16,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T01:18:00Z",
            "Maximum": 49.18,
            "Unit": "Percent"
        }
    ],
    "Label": "CPUUtilization"
}
```

Specifying multiple dimensions

The following example illustrates how to specify multiple dimensions. Each dimension is specified as a Name/Value pair, with a comma between the name and the value. Multiple dimensions are separated by a space. If a single metric includes multiple dimensions, you must specify a value for every defined dimension.

For more examples using the `get-metric-statistics` command, see Get Statistics for a Metric in the *Amazon CloudWatch Developer Guide*.

```
aws cloudwatch get-metric-statistics --metric-name Buffers --
namespace MyNameSpace --dimensions Name=InstanceID,Value=i-
abcdef Name=InstanceType,Value=m1.small --start-time 2016-10-15T04:00:00Z --end-
time 2016-10-19T07:00:00Z --statistics Average --period 60
```

- For API details, see GetMetricStatistics in *Amazon CLI Command Reference*.

Java

SDK for Java 2.x

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

```
    /**
     * Retrieves and displays metric statistics for the specified parameters.
     *
     * @param nameSpace    the namespace for the metric
     * @param metVal       the name of the metric
     * @param metricOption the statistic to retrieve for the metric (e.g.,
 "Maximum", "Average")
     * @param date         the date for which to retrieve the metric statistics,
 in the format "yyyy-MM-dd'T'HH:mm:ss'Z'"
     * @param myDimension  the dimension(s) to filter the metric statistics by
     * @return a {@link CompletableFuture} that completes when the metric
 statistics have been retrieved and displayed
     */
    public CompletableFuture<GetMetricStatisticsResponse>
 getAndDisplayMetricStatisticsAsync(String nameSpace, String metVal,

            String metricOption, String date, Dimension myDimension) {

        Instant start = Instant.parse(date);
        Instant endDate = Instant.now();

        // Building the request for metric statistics.
        GetMetricStatisticsRequest statisticsRequest =
 GetMetricStatisticsRequest.builder()
            .endTime(endDate)
            .startTime(start)
            .dimensions(myDimension)
            .metricName(metVal)
            .namespace(nameSpace)
            .period(86400) // 1 day period
            .statistics(Statistic.fromValue(metricOption))
            .build();
```

```
        return getAsyncClient().getMetricStatistics(statisticsRequest)
            .whenComplete((response, exception) -> {
                if (response != null) {
                    List<Datapoint> data = response.datapoints();
                    if (!data.isEmpty()) {
                        for (Datapoint datapoint : data) {
                            logger.info("Timestamp: {} Maximum value: {}",
 datapoint.timestamp(), datapoint.maximum());
                        }
                    } else {
                        logger.info("The returned data list is empty");
                    }
                } else {
                    logger.info("Failed to get metric statistics: {} ",
 exception.getMessage());
                }
            })
            .exceptionally(exception -> {
                throw new RuntimeException("Error while getting metric
 statistics: " + exception.getMessage(), exception);
            });
    }
```

- For API details, see GetMetricStatistics in *Amazon SDK for Java 2.x API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

```
suspend fun getAndDisplayMetricStatistics(
    nameSpaceVal: String,
    metVal: String,
    metricOption: String,
```

```
    date: String,
    myDimension: Dimension,
) {
    val start = Instant.parse(date)
    val endDate = Instant.now()
    val statisticsRequest =
        GetMetricStatisticsRequest {
            endTime =
                aws.smithy.kotlin.runtime.time
                    .Instant(endDate)
            startTime =
                aws.smithy.kotlin.runtime.time
                    .Instant(start)
            dimensions = listOf(myDimension)
            metricName = metVal
            namespace = nameSpaceVal
            period = 86400
            statistics = listOf(Statistic.fromValue(metricOption))
        }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        val response = cwClient.getMetricStatistics(statisticsRequest)
        val data = response.datapoints
        if (data != null) {
            if (data.isNotEmpty()) {
                for (datapoint in data) {
                    println("Timestamp: ${datapoint.timestamp} Maximum value:
 ${datapoint.maximum}")
                }
            } else {
                println("The returned data list is empty")
            }
        }
    }
}
```

- For API details, see GetMetricStatistics in *Amazon SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

```
class CloudWatchWrapper:
    """Encapsulates Amazon CloudWatch functions."""

    def __init__(self, cloudwatch_resource):
        """
        :param cloudwatch_resource: A Boto3 CloudWatch resource.
        """
        self.cloudwatch_resource = cloudwatch_resource


    def get_metric_statistics(self, namespace, name, start, end, period,
 stat_types):
        """
        Gets statistics for a metric within a specified time span. Metrics are
 grouped
        into the specified period.

        :param namespace: The namespace of the metric.
        :param name: The name of the metric.
        :param start: The UTC start time of the time span to retrieve.
        :param end: The UTC end time of the time span to retrieve.
        :param period: The period, in seconds, in which to group metrics. The
 period
                       must match the granularity of the metric, which depends on
                       the metric's age. For example, metrics that are older than
                       three hours have a one-minute granularity, so the period
 must
                       be at least 60 and must be a multiple of 60.
        :param stat_types: The type of statistics to retrieve, such as average
 value
                           or maximum value.
        :return: The retrieved statistics for the metric.
```

```
        """
        try:
            metric = self.cloudwatch_resource.Metric(namespace, name)
            stats = metric.get_statistics(
                StartTime=start, EndTime=end, Period=period,
 Statistics=stat_types
            )
            logger.info(
                "Got %s statistics for %s.", len(stats["Datapoints"]),
 stats["Label"]
            )
        except ClientError:
            logger.exception("Couldn't get statistics for %s.%s.", namespace,
 name)
            raise
        else:
            return stats
```

- For API details, see GetMetricStatistics in *Amazon SDK for Python (Boto3) API Reference*.

For a complete list of Amazon SDK developer guides and code examples, see Using CloudWatch with an Amazon SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `GetMetricWidgetImage` with an Amazon SDK or CLI

The following code examples show how to use `GetMetricWidgetImage`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Learn the basics

.NET

Amazon SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [Amazon Code Examples Repository](https://github.com/awsdocs/aws-doc-sdk-examples/tree/main/dotnetv3/CloudWatch#code-examples).

```
    /// <summary>
    /// Get an image for a metric graphed over time.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metric">The name of the metric.</param>
    /// <param name="stat">The name of the stat to chart.</param>
    /// <param name="period">The period to use for the chart.</param>
    /// <returns>A memory stream for the chart image.</returns>
    public async Task<MemoryStream> GetTimeSeriesMetricImage(string
 metricNamespace, string metric, string stat, int period)
    {
        var metricImageWidget = new
        {
            title = "Example Metric Graph",
            view = "timeSeries",
            stacked = false,
            period = period,
            width = 1400,
            height = 600,
            metrics = new List<List<object>>
                { new() { metricNamespace, metric, new { stat } } }
        };

        var metricImageWidgetString =
 JsonSerializer.Serialize(metricImageWidget);
        var imageResponse = await _amazonCloudWatch.GetMetricWidgetImageAsync(
            new GetMetricWidgetImageRequest()
            {
                MetricWidget = metricImageWidgetString
            });

        return imageResponse.MetricWidgetImage;
```

```
    }

    /// <summary>
    /// Save a metric image to a file.
    /// </summary>
    /// <param name="memoryStream">The MemoryStream for the metric image.</param>
    /// <param name="metricName">The name of the metric.</param>
    /// <returns>The path to the file.</returns>
    public string SaveMetricImage(MemoryStream memoryStream, string metricName)
    {
        var metricFileName = $"{metricName}_{DateTime.Now.Ticks}.png";
        using var sr = new StreamReader(memoryStream);
        // Writes the memory stream to a file.
        File.WriteAllBytes(metricFileName, memoryStream.ToArray());
        var filePath = Path.Join(AppDomain.CurrentDomain.BaseDirectory,
            metricFileName);
        return filePath;
    }
```

- For API details, see GetMetricWidgetImage in *Amazon SDK for .NET API Reference*.

CLI

Amazon CLI

To retrieve a snapshot graph of CPUUtilization

The following `get-metric-widget-image` example retrieves snapshot graph for the metric `CPUUtilization` of the EC2 instance with the ID `i-abcde` and saves the retrieved image as a file named "image.png" on your local machine.

```
aws cloudwatch get-metric-widget-image \
    --metric-widget '{"metrics":[["AWS/EC2","CPUUtilization","InstanceId","i-
abcde"]]}' \
    --output-format png \
    --output text | base64 --decode > image.png
```

This command produces no output.

- For API details, see GetMetricWidgetImage in *Amazon CLI Command Reference*.

Java

SDK for Java 2.x

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [Amazon Code Examples Repository](https://github.com/awsdocs/aws-doc-sdk-examples/tree/main/javav2/example_code/cloudwatch#code-examples).

```
    /**
     * Retrieves and saves a custom metric image to a file.
     *
     * @param fileName the name of the file to save the metric image to
     * @return a {@link CompletableFuture} that completes when the image has been
 saved to the file
     */
    public CompletableFuture<Void> downloadAndSaveMetricImageAsync(String
 fileName) {
        logger.info("Getting Image data for custom metric.");
        String myJSON = """
              {
                  "title": "Example Metric Graph",
                  "view": "timeSeries",
                  "stacked ": false,
                  "period": 10,
                  "width": 1400,
                  "height": 600,
                  "metrics": [
                      [
                      "AWS/Billing",
                      "EstimatedCharges",
                      "Currency",
                      "USD"
                     ]
                  ]
              }
             """;

        GetMetricWidgetImageRequest imageRequest =
 GetMetricWidgetImageRequest.builder()
```

```
            .metricWidget(myJSON)
            .build();

        return getAsyncClient().getMetricWidgetImage(imageRequest)
            .thenCompose(response -> {
                SdkBytes sdkBytes = response.metricWidgetImage();
                byte[] bytes = sdkBytes.asByteArray();
                return CompletableFuture.runAsync(() -> {
                    try {
                        File outputFile = new File(fileName);
                        try (FileOutputStream outputStream = new
 FileOutputStream(outputFile)) {
                            outputStream.write(bytes);
                        }
                    } catch (IOException e) {
                        throw new RuntimeException("Failed to write image to
 file", e);
                    }
                });
            })
            .whenComplete((result, exception) -> {
                if (exception != null) {
                    throw new RuntimeException("Error getting and saving metric
 image", exception);
                } else {
                    logger.info("Image data saved successfully to {}", fileName);
                }
            });
    }
```

- For API details, see GetMetricWidgetImage in *Amazon SDK for Java 2.x API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

```
suspend fun getAndOpenMetricImage(fileName: String) {
    println("Getting Image data for custom metric.")
    val myJSON = """{
        "title": "Example Metric Graph",
        "view": "timeSeries",
        "stacked ": false,
        "period": 10,
        "width": 1400,
        "height": 600,
        "metrics": [
            [
            "AWS/Billing",
            "EstimatedCharges",
            "Currency",
            "USD"
            ]
        ]
        }"""

    val imageRequest =
        GetMetricWidgetImageRequest {
            metricWidget = myJSON
        }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        val response = cwClient.getMetricWidgetImage(imageRequest)
        val bytes = response.metricWidgetImage
        if (bytes != null) {
            File(fileName).writeBytes(bytes)
        }
    }
    println("You have successfully written data to $fileName")
}
```

- For API details, see GetMetricWidgetImage in *Amazon SDK for Kotlin API reference*.

For a complete list of Amazon SDK developer guides and code examples, see Using CloudWatch with an Amazon SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `ListDashboards` with an Amazon SDK or CLI

The following code examples show how to use `ListDashboards`.

.NET

Amazon SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [Amazon Code Examples Repository](https://github.com/awsdocs/aws-doc-sdk-examples/tree/main/dotnetv3/CloudWatch#code-examples).

```
/// <summary>
/// Get a list of dashboards.
/// </summary>
/// <returns>A list of DashboardEntry objects.</returns>
public async Task<List<DashboardEntry>> ListDashboards()
{
    var results = new List<DashboardEntry>();
    var paginateDashboards = _amazonCloudWatch.Paginators.ListDashboards(
        new ListDashboardsRequest());
    // Get the entire list using the paginator.
    await foreach (var data in paginateDashboards.DashboardEntries)
    {
        results.Add(data);
    }

    return results;
}
```

- For API details, see ListDashboards in *Amazon SDK for .NET API Reference*.

CLI

Amazon CLI

To retrieve a list of Dashboards

The following `list-dashboards` example lists all the Dashboards in the specified account.

```
aws cloudwatch list-dashboards
```

Output:

```
{
    "DashboardEntries": [
        {
            "DashboardName": "Dashboard-A",
            "DashboardArn": "arn:aws:cloudwatch::123456789012:dashboard/
Dashboard-A",
            "LastModified": "2024-10-11T18:40:11+00:00",
            "Size": 271
        },
        {
            "DashboardName": "Dashboard-B",
            "DashboardArn": "arn:aws:cloudwatch::123456789012:dashboard/
Dashboard-B",
            "LastModified": "2024-10-11T18:44:41+00:00",
            "Size": 522
        }
    ]
}
```

For more information, see Amazon CloudWatch dashboards in the *Amazon CloudWatch User Guide*.

- For API details, see ListDashboards in *Amazon CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

```
    /**
     * Lists the available dashboards.
```

```
     *
     * @return a {@link CompletableFuture} that completes when the operation is
finished.
     * The future will complete exceptionally if an error occurs while listing
the dashboards.
     */
    public CompletableFuture<Void> listDashboardsAsync() {
        ListDashboardsRequest listDashboardsRequest =
ListDashboardsRequest.builder().build();
        ListDashboardsPublisher paginator =
getAsyncClient().listDashboardsPaginator(listDashboardsRequest);
        return paginator.subscribe(response -> {
            response.dashboardEntries().forEach(entry -> {
                logger.info("Dashboard name is: {} ", entry.dashboardName());
                logger.info("Dashboard ARN is: {} ", entry.dashboardArn());
            });
        }).exceptionally(ex -> {
            logger.info("Failed to list dashboards: {} ", ex.getMessage());
            throw new RuntimeException("Error occurred while listing dashboards",
ex);
        });
    }
```

- For API details, see ListDashboards in *Amazon SDK for Java 2.x API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

```
suspend fun listDashboards() {
    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient
            .listDashboardsPaginated({})
            .transform { it.dashboardEntries?.forEach { obj -> emit(obj) } }
            .collect { obj ->
```

```
                println("Name is ${obj.dashboardName}")
                println("Dashboard ARN is ${obj.dashboardArn}")
            }
        }
    }
```

- For API details, see ListDashboards in *Amazon SDK for Kotlin API reference*.

PowerShell

Tools for PowerShell

Example 1: Returns the collection of dashboards for your account.

```
Get-CWDashboardList
```

Output:

```
DashboardArn DashboardName LastModified        Size
------------ ------------- ------------        ----
arn:...      Dashboard1    7/6/2017 8:14:15 PM 252
```

Example 2: Returns the collection of dashboards for your account whose names start with the prefix 'dev'.

```
Get-CWDashboardList -DashboardNamePrefix dev
```

- For API details, see ListDashboards in *Amazon Tools for PowerShell Cmdlet Reference*.

For a complete list of Amazon SDK developer guides and code examples, see Using CloudWatch with an Amazon SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `ListMetrics` with an Amazon SDK or CLI

The following code examples show how to use `ListMetrics`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Learn the basics
- Manage metrics and alarms

.NET

Amazon SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

```
    /// <summary>
    /// List metrics available, optionally within a namespace.
    /// </summary>
    /// <param name="metricNamespace">Optional CloudWatch namespace to use when
listing metrics.</param>
    /// <param name="filter">Optional dimension filter.</param>
    /// <param name="metricName">Optional metric name filter.</param>
    /// <returns>The list of metrics.</returns>
    public async Task<List<Metric>> ListMetrics(string? metricNamespace = null,
DimensionFilter? filter = null, string? metricName = null)
    {
        var results = new List<Metric>();
        var paginateMetrics = _amazonCloudWatch.Paginators.ListMetrics(
            new ListMetricsRequest
            {
                Namespace = metricNamespace,
                Dimensions = filter != null ? new List<DimensionFilter>
{ filter } : null,
                MetricName = metricName
            });
        // Get the entire list using the paginator.
        await foreach (var metric in paginateMetrics.Metrics)
        {
            results.Add(metric);
        }

        return results;
    }
```

- For API details, see ListMetrics in *Amazon SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

Include the required files.

```
#include <aws/core/Aws.h>
#include <aws/monitoring/CloudWatchClient.h>
#include <aws/monitoring/model/ListMetricsRequest.h>
#include <aws/monitoring/model/ListMetricsResult.h>
#include <iomanip>
#include <iostream>
```

List the metrics.

```
        Aws::CloudWatch::CloudWatchClient cw;
        Aws::CloudWatch::Model::ListMetricsRequest request;

        if (argc > 1)
        {
            request.SetMetricName(argv[1]);
        }

        if (argc > 2)
        {
            request.SetNamespace(argv[2]);
        }

        bool done = false;
        bool header = false;
```

```
        while (!done)
        {
            auto outcome = cw.ListMetrics(request);
            if (!outcome.IsSuccess())
            {
                std::cout << "Failed to list CloudWatch metrics:" <<
                    outcome.GetError().GetMessage() << std::endl;
                break;
            }

            if (!header)
            {
                std::cout << std::left << std::setw(48) << "MetricName" <<
                    std::setw(32) << "Namespace" << "DimensionNameValuePairs" <<
                    std::endl;
                header = true;
            }

            const auto &metrics = outcome.GetResult().GetMetrics();
            for (const auto &metric : metrics)
            {
                std::cout << std::left << std::setw(48) <<
                    metric.GetMetricName() << std::setw(32) <<
                    metric.GetNamespace();
                const auto &dimensions = metric.GetDimensions();
                for (auto iter = dimensions.cbegin();
                    iter != dimensions.cend(); ++iter)
                {
                    const auto &dimkv = *iter;
                    std::cout << dimkv.GetName() << " = " << dimkv.GetValue();
                    if (iter + 1 != dimensions.cend())
                    {
                        std::cout << ", ";
                    }
                }
                std::cout << std::endl;
            }

            const auto &next_token = outcome.GetResult().GetNextToken();
            request.SetNextToken(next_token);
            done = next_token.empty();
        }
```

- For API details, see ListMetrics in *Amazon SDK for C++ API Reference*.

CLI

Amazon CLI

To list the metrics for Amazon SNS

The following `list-metrics` example displays the metrics for Amazon SNS.

```
aws cloudwatch list-metrics \
    --namespace "AWS/SNS"
```

Output:

```
{
    "Metrics": [
        {
            "Namespace": "AWS/SNS",
            "Dimensions": [
                {
                    "Name": "TopicName",
                    "Value": "NotifyMe"
                }
            ],
            "MetricName": "PublishSize"
        },
        {
            "Namespace": "AWS/SNS",
            "Dimensions": [
                {
                    "Name": "TopicName",
                    "Value": "CFO"
                }
            ],
            "MetricName": "PublishSize"
        },
        {
            "Namespace": "AWS/SNS",
            "Dimensions": [
                {
                    "Name": "TopicName",
                    "Value": "NotifyMe"
```

```
                }
            ],
            "MetricName": "NumberOfNotificationsFailed"
        },
        {
            "Namespace": "AWS/SNS",
            "Dimensions": [
                {
                    "Name": "TopicName",
                    "Value": "NotifyMe"
                }
            ],
            "MetricName": "NumberOfNotificationsDelivered"
        },
        {
            "Namespace": "AWS/SNS",
            "Dimensions": [
                {
                    "Name": "TopicName",
                    "Value": "NotifyMe"
                }
            ],
            "MetricName": "NumberOfMessagesPublished"
        },
        {
            "Namespace": "AWS/SNS",
            "Dimensions": [
                {
                    "Name": "TopicName",
                    "Value": "CFO"
                }
            ],
            "MetricName": "NumberOfMessagesPublished"
        },
        {
            "Namespace": "AWS/SNS",
            "Dimensions": [
                {
                    "Name": "TopicName",
                    "Value": "CFO"
                }
            ],
            "MetricName": "NumberOfNotificationsDelivered"
        },
```

```
        {
            "Namespace": "AWS/SNS",
            "Dimensions": [
                {
                    "Name": "TopicName",
                    "Value": "CFO"
                }
            ],
            "MetricName": "NumberOfNotificationsFailed"
        }
    ]
}
```

- For API details, see ListMetrics in *Amazon CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

```
    /**
     * Retrieves a list of metric names for the specified namespace.
     *
     * @param namespace the namespace for which to retrieve the metric names
     * @return a {@link CompletableFuture} that, when completed, contains an
 {@link ArrayList} of
     * the metric names in the specified namespace
     * @throws RuntimeException if an error occurs while listing the metrics
     */
    public CompletableFuture<ArrayList<String>> listMetsAsync(String namespace) {
        ListMetricsRequest request = ListMetricsRequest.builder()
            .namespace(namespace)
            .build();

        ListMetricsPublisher metricsPaginator =
 getAsyncClient().listMetricsPaginator(request);
        Set<String> metSet = new HashSet<>();
```

```
        CompletableFuture<Void> future = metricsPaginator.subscribe(response -> {
            response.metrics().forEach(metric -> {
                String metricName = metric.metricName();
                metSet.add(metricName);
            });
        });

        return future
            .thenApply(ignored -> new ArrayList<>(metSet))
            .exceptionally(exception -> {
                throw new RuntimeException("Failed to list metrics: " +
 exception.getMessage(), exception);
            });
    }
```

- For API details, see ListMetrics in *Amazon SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

Import the SDK and client modules and call the API.

```
import {
  CloudWatchServiceException,
  ListMetricsCommand,
} from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

export const main = async () => {
  // Use the AWS console to see available namespaces and metric names. Custom
 metrics can also be created.
  // https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/
viewing_metrics_with_cloudwatch.html
  const command = new ListMetricsCommand({
```

```
    Dimensions: [
      {
        Name: "LogGroupName",
      },
    ],
    MetricName: "IncomingLogEvents",
    Namespace: "AWS/Logs",
  });

  try {
    const response = await client.send(command);
    console.log(`Metrics count: ${response.Metrics?.length}`);
    return response;
  } catch (caught) {
    if (caught instanceof CloudWatchServiceException) {
      console.error(`Error from CloudWatch. ${caught.name}: ${caught.message}`);
    } else {
      throw caught;
    }
  }
};
```

Create the client in a separate module and export it.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- For more information, see Amazon SDK for JavaScript Developer Guide.
- For API details, see ListMetrics in *Amazon SDK for JavaScript API Reference*.

SDK for JavaScript (v2)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

```
// Load the AWS SDK for Node.js
```

```
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatch service object
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });

var params = {
  Dimensions: [
    {
      Name: "LogGroupName" /* required */,
    },
  ],
  MetricName: "IncomingLogEvents",
  Namespace: "AWS/Logs",
};

cw.listMetrics(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Metrics", JSON.stringify(data.Metrics));
  }
});
```

- For more information, see Amazon SDK for JavaScript Developer Guide.
- For API details, see ListMetrics in *Amazon SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

```
suspend fun listMets(namespaceVal: String?): ArrayList<String>? {
    val metList = ArrayList<String>()
```

```
    val request =
        ListMetricsRequest {
            namespace = namespaceVal
        }
    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        val reponse = cwClient.listMetrics(request)
        reponse.metrics?.forEach { metrics ->
            val data = metrics.metricName
            if (!metList.contains(data)) {
                metList.add(data!!)
            }
        }
    }
    return metList
}
```

- For API details, see [ListMetrics](#) in *Amazon SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [Amazon Code Examples Repository](#).

```
class CloudWatchWrapper:
    """Encapsulates Amazon CloudWatch functions."""

    def __init__(self, cloudwatch_resource):
        """
        :param cloudwatch_resource: A Boto3 CloudWatch resource.
        """
        self.cloudwatch_resource = cloudwatch_resource


    def list_metrics(self, namespace, name, recent=False):
        """
        Gets the metrics within a namespace that have the specified name.
```

```
        If the metric has no dimensions, a single metric is returned.
        Otherwise, metrics for all dimensions are returned.

        :param namespace: The namespace of the metric.
        :param name: The name of the metric.
        :param recent: When True, only metrics that have been active in the last
                       three hours are returned.
        :return: An iterator that yields the retrieved metrics.
        """
        try:
            kwargs = {"Namespace": namespace, "MetricName": name}
            if recent:
                kwargs["RecentlyActive"] = "PT3H"  # List past 3 hours only
            metric_iter = self.cloudwatch_resource.metrics.filter(**kwargs)
            logger.info("Got metrics for %s.%s.", namespace, name)
        except ClientError:
            logger.exception("Couldn't get metrics for %s.%s.", namespace, name)
            raise
        else:
            return metric_iter
```

- For API details, see ListMetrics in *Amazon SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

```
# Lists available metrics for a metric namespace in Amazon CloudWatch.
#
# @param cloudwatch_client [Aws::CloudWatch::Client]
#   An initialized CloudWatch client.
# @param metric_namespace [String] The namespace of the metric.
# @example
#   list_metrics_for_namespace(
```

```
#     Aws::CloudWatch::Client.new(region: 'us-east-1'),
#     'SITE/TRAFFIC'
#   )
def list_metrics_for_namespace(cloudwatch_client, metric_namespace)
  response = cloudwatch_client.list_metrics(namespace: metric_namespace)

  if response.metrics.count.positive?
    response.metrics.each do |metric|
      puts "  Metric name: #{metric.metric_name}"
      if metric.dimensions.count.positive?
        puts '    Dimensions:'
        metric.dimensions.each do |dimension|
          puts "      Name: #{dimension.name}, Value: #{dimension.value}"
        end
      else
        puts 'No dimensions found.'
      end
    end
  else
    puts "No metrics found for namespace '#{metric_namespace}'. " \
      'Note that it could take up to 15 minutes for recently-added metrics ' \
      'to become available.'
  end
end

# Example usage:
def run_me
  metric_namespace = 'SITE/TRAFFIC'
  # Replace us-west-2 with the AWS Region you're using for Amazon CloudWatch.
  region = 'us-east-1'

  cloudwatch_client = Aws::CloudWatch::Client.new(region: region)

  # Add three datapoints.
  puts 'Continuing...' unless datapoint_added_to_metric?(
    cloudwatch_client,
    metric_namespace,
    'UniqueVisitors',
    'SiteName',
    'example.com',
    5_885.0,
    'Count'
  )
```

```
  puts 'Continuing...' unless datapoint_added_to_metric?(
    cloudwatch_client,
    metric_namespace,
    'UniqueVisits',
    'SiteName',
    'example.com',
    8_628.0,
    'Count'
  )

  puts 'Continuing...' unless datapoint_added_to_metric?(
    cloudwatch_client,
    metric_namespace,
    'PageViews',
    'PageURL',
    'example.html',
    18_057.0,
    'Count'
  )

  puts "Metrics for namespace '#{metric_namespace}':"
  list_metrics_for_namespace(cloudwatch_client, metric_namespace)
end

run_me if $PROGRAM_NAME == __FILE__
```

- For API details, see ListMetrics in *Amazon SDK for Ruby API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

```
    "The following list-metrics example displays the metrics for Amazon
 CloudWatch."
    TRY.
```

```
        oo_result = lo_cwt->listmetrics(            " oo_result is returned for
testing purposes. "
          iv_namespace = iv_namespace ).
        DATA(lt_metrics) = oo_result->get_metrics( ).
        MESSAGE 'Metrics retrieved.' TYPE 'I'.
      CATCH /aws1/cx_cwtinvparamvalueex.
        MESSAGE 'The specified argument was not valid.' TYPE 'E'.
    ENDTRY.
```

- For API details, see ListMetrics in *Amazon SDK for SAP ABAP API reference*.

For a complete list of Amazon SDK developer guides and code examples, see Using CloudWatch with an Amazon SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `PutAnomalyDetector` with an Amazon SDK or CLI

The following code examples show how to use `PutAnomalyDetector`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Learn the basics

.NET

Amazon SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

```
    /// <summary>
    /// Add an anomaly detector for a single metric.
    /// </summary>
    /// <param name="anomalyDetector">A single metric anomaly detector.</param>
    /// <returns>True if successful.</returns>
```

```
    public async Task<bool> PutAnomalyDetector(SingleMetricAnomalyDetector
 anomalyDetector)
    {
        var putAlarmDetectorResult = await
 _amazonCloudWatch.PutAnomalyDetectorAsync(
            new PutAnomalyDetectorRequest()
            {
                SingleMetricAnomalyDetector = anomalyDetector
            });

        return putAlarmDetectorResult.HttpStatusCode == HttpStatusCode.OK;
    }
```

- For API details, see PutAnomalyDetector in *Amazon SDK for .NET API Reference*.

CLI

Amazon CLI

To create an anomaly detection model

The following `put-anomaly-detector` example creates an anomaly detection model for a CloudWatch metric.

```
aws cloudwatch put-anomaly-detector \
    --namespace AWS/Logs \
    --metric-name IncomingBytes \
    --stat SampleCount
```

This command produces no output.

For more information, see Using CloudWatch anomaly detection in the *Amazon CloudWatch User Guide*.

- For API details, see PutAnomalyDetector in *Amazon CLI Command Reference*.

Java

SDK for Java 2.x

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [Amazon Code Examples Repository](https://github.com/awsdocs/aws-doc-sdk-examples/tree/main/javav2/example_code/cloudwatch#code-examples).

```
    /**
     * Adds an anomaly detector for the given file.
     *
     * @param fileName the name of the file containing the anomaly detector
 configuration
     * @return a {@link CompletableFuture} that completes when the anomaly
 detector has been added
     */
    public CompletableFuture<Void> addAnomalyDetectorAsync(String fileName) {
        CompletableFuture<JsonNode> readFileFuture =
 CompletableFuture.supplyAsync(() -> {
            try {
                JsonParser parser = new JsonFactory().createParser(new
 File(fileName));
                return new ObjectMapper().readTree(parser); // Return the root
 node
            } catch (IOException e) {
                throw new RuntimeException("Failed to read or parse the file",
 e);
            }
        });

        return readFileFuture.thenCompose(rootNode -> {
            try {
                String customMetricNamespace =
 rootNode.findValue("customMetricNamespace").asText();
                String customMetricName =
 rootNode.findValue("customMetricName").asText();

                SingleMetricAnomalyDetector singleMetricAnomalyDetector =
 SingleMetricAnomalyDetector.builder()
```

```
                    .metricName(customMetricName)
                    .namespace(customMetricNamespace)
                    .stat("Maximum")
                    .build();

                PutAnomalyDetectorRequest anomalyDetectorRequest =
PutAnomalyDetectorRequest.builder()
                    .singleMetricAnomalyDetector(singleMetricAnomalyDetector)
                    .build();

                return
getAsyncClient().putAnomalyDetector(anomalyDetectorRequest).thenAccept(response
-> {
                    logger.info("Added anomaly detector for metric {}",
customMetricName);
                });
            } catch (Exception e) {
                throw new RuntimeException("Failed to create anomaly detector",
e);
            }
        }).whenComplete((result, exception) -> {
            if (exception != null) {
                throw new RuntimeException("Error adding anomaly detector",
exception);
            }
        });
    }
```

- For API details, see [PutAnomalyDetector](#) in *Amazon SDK for Java 2.x API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [Amazon Code Examples Repository](#).

```
suspend fun addAnomalyDetector(fileName: String?) {
```

```
    // Read values from the JSON file.
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val customMetricNamespace =
 rootNode.findValue("customMetricNamespace").asText()
    val customMetricName = rootNode.findValue("customMetricName").asText()

    val singleMetricAnomalyDetectorVal =
        SingleMetricAnomalyDetector {
            metricName = customMetricName
            namespace = customMetricNamespace
            stat = "Maximum"
        }

    val anomalyDetectorRequest =
        PutAnomalyDetectorRequest {
            singleMetricAnomalyDetector = singleMetricAnomalyDetectorVal
        }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient.putAnomalyDetector(anomalyDetectorRequest)
        println("Added anomaly detector for metric $customMetricName.")
    }
}
```

- For API details, see PutAnomalyDetector in *Amazon SDK for Kotlin API reference*.

For a complete list of Amazon SDK developer guides and code examples, see Using CloudWatch with an Amazon SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `PutDashboard` with an Amazon SDK or CLI

The following code examples show how to use `PutDashboard`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Learn the basics

.NET

Amazon SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

```
    /// <summary>
    /// Set up a dashboard using a call to the wrapper class.
    /// </summary>
    /// <param name="customMetricNamespace">The metric namespace.</param>
    /// <param name="customMetricName">The metric name.</param>
    /// <param name="dashboardName">The name of the dashboard.</param>
    /// <returns>A list of validation messages.</returns>
    private static async Task<List<DashboardValidationMessage>> SetupDashboard(
        string customMetricNamespace, string customMetricName, string
 dashboardName)
    {
        // Get the dashboard model from configuration.
        var newDashboard = new DashboardModel();
        _configuration.GetSection("dashboardExampleBody").Bind(newDashboard);

        // Add a new metric to the dashboard.
        newDashboard.Widgets.Add(new Widget
        {
            Height = 8,
            Width = 8,
            Y = 8,
            X = 0,
            Type = "metric",
            Properties = new Properties
            {
                Metrics = new List<List<object>>
                    { new() { customMetricNamespace, customMetricName } },
                View = "timeSeries",
                Region = "us-east-1",
                Stat = "Sum",
                Period = 86400,
```

```
                YAxis = new YAxis { Left = new Left { Min = 0, Max = 100 } },
                Title = "Custom Metric Widget",
                LiveData = true,
                Sparkline = true,
                Trend = true,
                Stacked = false,
                SetPeriodToTimeRange = false
            }
        });

        var newDashboardString = JsonSerializer.Serialize(newDashboard,
            new JsonSerializerOptions
            { DefaultIgnoreCondition = JsonIgnoreCondition.WhenWritingNull });
        var validationMessages =
            await _cloudWatchWrapper.PutDashboard(dashboardName,
 newDashboardString);

        return validationMessages;
    }

    /// <summary>
    /// Wrapper to create or add to a dashboard with metrics.
    /// </summary>
    /// <param name="dashboardName">The name for the dashboard.</param>
    /// <param name="dashboardBody">The metric data in JSON for the dashboard.</
param>
    /// <returns>A list of validation messages for the dashboard.</returns>
    public async Task<List<DashboardValidationMessage>> PutDashboard(string
 dashboardName,
        string dashboardBody)
    {
        // Updating a dashboard replaces all contents.
        // Best practice is to include a text widget indicating this dashboard
 was created programmatically.
        var dashboardResponse = await _amazonCloudWatch.PutDashboardAsync(
            new PutDashboardRequest()
            {
                DashboardName = dashboardName,
                DashboardBody = dashboardBody
            });

        return dashboardResponse.DashboardValidationMessages;
    }
```

- For API details, see PutDashboard in *Amazon SDK for .NET API Reference*.

CLI

Amazon CLI

To create a dashboard

The following `put-dashboard` example creates a dashboard named `Dashboard-A` in the specified account.

```
aws cloudwatch put-dashboard \
    --dashboard-name Dashboard-A \
    --dashboard-body '{"widgets":
[{"height":6,"width":6,"y":0,"x":0,"type":"metric","properties":
{"view":"timeSeries","stacked":false,"metrics":
[["Namespace","CPUUtilization","Environment","Prod","Type","App"]],"region":"us-
east-1"}}]}'
```

Output:

```
{
    "DashboardValidationMessages": []
}
```

For more information, see Creating a CloudWatch dashboard in the *Amazon CloudWatch User Guide*.

- For API details, see PutDashboard in *Amazon CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

```
    /**
     * Creates a new dashboard with the specified name and metrics from the given
file.
     *
     * @param dashboardName the name of the dashboard to be created
     * @param fileName      the name of the file containing the dashboard body
     * @return a {@link CompletableFuture} representing the asynchronous
operation of creating the dashboard
     * @throws IOException if there is an error reading the dashboard body from
the file
     */
    public CompletableFuture<PutDashboardResponse>
createDashboardWithMetricsAsync(String dashboardName, String fileName) throws
IOException {
        String dashboardBody = readFileAsString(fileName);
        PutDashboardRequest dashboardRequest = PutDashboardRequest.builder()
            .dashboardName(dashboardName)
            .dashboardBody(dashboardBody)
            .build();

        return getAsyncClient().putDashboard(dashboardRequest)
            .handle((response, ex) -> {
                if (ex != null) {
                    logger.info("Failed to create dashboard: {}",
ex.getMessage());
                    throw new RuntimeException("Dashboard creation failed", ex);
                } else {
                    // Handle the normal response case
                    logger.info("{} was successfully created.", dashboardName);
                    List<DashboardValidationMessage> messages =
response.dashboardValidationMessages();
                    if (messages.isEmpty()) {
                        logger.info("There are no messages in the new
Dashboard.");
                    } else {
                        for (DashboardValidationMessage message : messages) {
                            logger.info("Message: {}", message.message());
                        }
                    }
                    return response; // Return the response for further use
                }
            });
```

```
    }
```

- For API details, see PutDashboard in *Amazon SDK for Java 2.x API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

```
suspend fun createDashboardWithMetrics(
    dashboardNameVal: String,
    fileNameVal: String,
) {
    val dashboardRequest =
        PutDashboardRequest {
            dashboardName = dashboardNameVal
            dashboardBody = readFileAsString(fileNameVal)
        }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        val response = cwClient.putDashboard(dashboardRequest)
        println("$dashboardNameVal was successfully created.")
        val messages = response.dashboardValidationMessages
        if (messages != null) {
            if (messages.isEmpty()) {
                println("There are no messages in the new Dashboard")
            } else {
                for (message in messages) {
                    println("Message is: ${message.message}")
                }
            }
        }
    }
}
```

- For API details, see [PutDashboard](#) in *Amazon SDK for Kotlin API reference*.

PowerShell

Tools for PowerShell

Example 1: Creates or updates the dashboard named 'Dashboard1' to include two metric widgets side by side.

```
$dashBody = @"
{
    "widgets":[
        {
             "type":"metric",
             "x":0,
             "y":0,
             "width":12,
             "height":6,
             "properties":{
                "metrics":[
                   [
                      "AWS/EC2",
                      "CPUUtilization",
                      "InstanceId",
                      "i-012345"
                   ]
                ],
                "period":300,
                "stat":"Average",
                "region":"us-east-1",
                "title":"EC2 Instance CPU"
             }
        },
        {
             "type":"metric",
             "x":12,
             "y":0,
             "width":12,
             "height":6,
             "properties":{
                "metrics":[
                   [
                      "AWS/S3",
```

```
                        "BucketSizeBytes",
                        "BucketName",
                        "amzn-s3-demo-bucket"
                     ]
                  ],
                  "period":86400,
                  "stat":"Maximum",
                  "region":"us-east-1",
                  "title":"amzn-s3-demo-bucket bytes"
              }
          }
      ]
  }
  "@

  Write-CWDashboard -DashboardName Dashboard1 -DashboardBody $dashBody
```

Example 2: Creates or updates the dashboard, piping the content describing the dashboard into the cmdlet.

```
$dashBody = @"
{
...
}
"@

$dashBody | Write-CWDashboard -DashboardName Dashboard1
```

- For API details, see PutDashboard in *Amazon Tools for PowerShell Cmdlet Reference*.

For a complete list of Amazon SDK developer guides and code examples, see Using CloudWatch with an Amazon SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `PutMetricAlarm` with an Amazon SDK or CLI

The following code examples show how to use `PutMetricAlarm`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Learn the basics

- Get started with alarms
- Manage metrics and alarms

.NET

Amazon SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

```
    /// <summary>
    /// Add a metric alarm to send an email when the metric passes a threshold.
    /// </summary>
    /// <param name="alarmDescription">A description of the alarm.</param>
    /// <param name="alarmName">The name for the alarm.</param>
    /// <param name="comparison">The type of comparison to use.</param>
    /// <param name="metricName">The name of the metric for the alarm.</param>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="threshold">The threshold value for the alarm.</param>
    /// <param name="alarmActions">Optional actions to execute when in an alarm
 state.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> PutMetricEmailAlarm(string alarmDescription, string
 alarmName, ComparisonOperator comparison,
        string metricName, string metricNamespace, double threshold, List<string>
 alarmActions = null!)
    {
        try
        {
            var putEmailAlarmResponse = await
 _amazonCloudWatch.PutMetricAlarmAsync(
                new PutMetricAlarmRequest()
                {
                    AlarmActions = alarmActions,
                    AlarmDescription = alarmDescription,
                    AlarmName = alarmName,
                    ComparisonOperator = comparison,
                    Threshold = threshold,
```

```
                        Namespace = metricNamespace,
                        MetricName = metricName,
                        EvaluationPeriods = 1,
                        Period = 10,
                        Statistic = new Statistic("Maximum"),
                        DatapointsToAlarm = 1,
                        TreatMissingData = "ignore"
                    });
                return putEmailAlarmResponse.HttpStatusCode == HttpStatusCode.OK;
            }
            catch (LimitExceededException lex)
            {
                _logger.LogError(lex, $"Unable to add alarm {alarmName}. Alarm quota
 has already been reached.");
            }

            return false;
        }

        /// <summary>
        /// Add specific email actions to a list of action strings for a CloudWatch
 alarm.
        /// </summary>
        /// <param name="accountId">The AccountId for the alarm.</param>
        /// <param name="region">The region for the alarm.</param>
        /// <param name="emailTopicName">An Amazon Simple Notification Service (SNS)
 topic for the alarm email.</param>
        /// <param name="alarmActions">Optional list of existing alarm actions to
 append to.</param>
        /// <returns>A list of string actions for an alarm.</returns>
        public List<string> AddEmailAlarmAction(string accountId, string region,
            string emailTopicName, List<string>? alarmActions = null)
        {
            alarmActions ??= new List<string>();
            var snsAlarmAction = $"arn:aws:sns:{region}:{accountId}:
{emailTopicName}";
            alarmActions.Add(snsAlarmAction);
            return alarmActions;
        }
```

- For API details, see PutMetricAlarm in *Amazon SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [Amazon Code Examples Repository](#).

Include the required files.

```
#include <aws/core/Aws.h>
#include <aws/monitoring/CloudWatchClient.h>
#include <aws/monitoring/model/PutMetricAlarmRequest.h>
#include <iostream>
```

Create the alarm to watch the metric.

```
        Aws::CloudWatch::CloudWatchClient cw;
        Aws::CloudWatch::Model::PutMetricAlarmRequest request;
        request.SetAlarmName(alarm_name);
        request.SetComparisonOperator(
            Aws::CloudWatch::Model::ComparisonOperator::GreaterThanThreshold);
        request.SetEvaluationPeriods(1);
        request.SetMetricName("CPUUtilization");
        request.SetNamespace("AWS/EC2");
        request.SetPeriod(60);
        request.SetStatistic(Aws::CloudWatch::Model::Statistic::Average);
        request.SetThreshold(70.0);
        request.SetActionsEnabled(false);
        request.SetAlarmDescription("Alarm when server CPU exceeds 70%");
        request.SetUnit(Aws::CloudWatch::Model::StandardUnit::Seconds);

        Aws::CloudWatch::Model::Dimension dimension;
        dimension.SetName("InstanceId");
        dimension.SetValue(instanceId);

        request.AddDimensions(dimension);

        auto outcome = cw.PutMetricAlarm(request);
```

```
        if (!outcome.IsSuccess())
        {
            std::cout << "Failed to create CloudWatch alarm:" <<
                outcome.GetError().GetMessage() << std::endl;
        }
        else
        {
            std::cout << "Successfully created CloudWatch alarm " << alarm_name
                << std::endl;
        }
```

- For API details, see PutMetricAlarm in *Amazon SDK for C++ API Reference*.

CLI

Amazon CLI

To send an Amazon Simple Notification Service email message when CPU utilization exceeds 70 percent

The following example uses the `put-metric-alarm` command to send an Amazon Simple Notification Service email message when CPU utilization exceeds 70 percent:

```
aws cloudwatch put-metric-alarm --alarm-name cpu-mon --alarm-description "Alarm
 when CPU exceeds 70 percent" --metric-name CPUUtilization --namespace AWS/
EC2 --statistic Average --period 300 --threshold 70 --comparison-
operator GreaterThanThreshold  --dimensions "Name=InstanceId,Value=i-12345678" --
evaluation-periods 2 --alarm-actions arn:aws:sns:us-east-1:111122223333:MyTopic
 --unit Percent
```

This command returns to the prompt if successful. If an alarm with the same name already exists, it will be overwritten by the new alarm.

To specify multiple dimensions

The following example illustrates how to specify multiple dimensions. Each dimension is specified as a Name/Value pair, with a comma between the name and the value. Multiple dimensions are separated by a space:

```
aws cloudwatch put-metric-alarm --alarm-name "Default_Test_Alarm3" --alarm-
description "The default example alarm" --namespace "CW EXAMPLE METRICS"
```

```
--metric-name Default_Test --statistic Average --period 60 --evaluation-
periods 3 --threshold 50 --comparison-operator GreaterThanOrEqualToThreshold --
dimensions Name=key1,Value=value1 Name=key2,Value=value2
```

- For API details, see PutMetricAlarm in *Amazon CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

```
    /**
     * Creates an alarm based on the configuration provided in a JSON file.
     *
     * @param fileName the name of the JSON file containing the alarm
 configuration
     * @return a CompletableFuture that represents the asynchronous operation of
 creating the alarm
     * @throws RuntimeException if an exception occurs while reading the JSON
 file or creating the alarm
     */
    public CompletableFuture<String> createAlarmAsync(String fileName) {
        com.fasterxml.jackson.databind.JsonNode rootNode;
        try {
            JsonParser parser = new JsonFactory().createParser(new
 File(fileName));
            rootNode = new ObjectMapper().readTree(parser);
        } catch (IOException e) {
            throw new RuntimeException("Failed to read the alarm configuration
 file", e);
        }

        // Extract values from the JSON node.
        String customMetricNamespace =
 rootNode.findValue("customMetricNamespace").asText();
        String customMetricName =
 rootNode.findValue("customMetricName").asText();
```

```
        String alarmName = rootNode.findValue("exampleAlarmName").asText();
        String emailTopic = rootNode.findValue("emailTopic").asText();
        String accountId = rootNode.findValue("accountId").asText();
        String region = rootNode.findValue("region").asText();

        // Create a List for alarm actions.
        List<String> alarmActions = new ArrayList<>();
        alarmActions.add("arn:aws:sns:" + region + ":" + accountId + ":" +
emailTopic);

        PutMetricAlarmRequest alarmRequest = PutMetricAlarmRequest.builder()
            .alarmActions(alarmActions)
            .alarmDescription("Example metric alarm")
            .alarmName(alarmName)

.comparisonOperator(ComparisonOperator.GREATER_THAN_OR_EQUAL_TO_THRESHOLD)
            .threshold(100.00)
            .metricName(customMetricName)
            .namespace(customMetricNamespace)
            .evaluationPeriods(1)
            .period(10)
            .statistic("Maximum")
            .datapointsToAlarm(1)
            .treatMissingData("ignore")
            .build();

        // Call the putMetricAlarm asynchronously and handle the result.
        return getAsyncClient().putMetricAlarm(alarmRequest)
            .handle((response, ex) -> {
                if (ex != null) {
                    logger.info("Failed to create alarm: {}", ex.getMessage());
                    throw new RuntimeException("Failed to create alarm", ex);
                } else {
                    logger.info("{} was successfully created!", alarmName);
                    return alarmName;
                }
            });
    }
```

- For API details, see PutMetricAlarm in *Amazon SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

Import the SDK and client modules and call the API.

```
import { PutMetricAlarmCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

const run = async () => {
  // This alarm triggers when CPUUtilization exceeds 70% for one minute.
  const command = new PutMetricAlarmCommand({
    AlarmName: process.env.CLOUDWATCH_ALARM_NAME, // Set the value of
 CLOUDWATCH_ALARM_NAME to the name of an existing alarm.
    ComparisonOperator: "GreaterThanThreshold",
    EvaluationPeriods: 1,
    MetricName: "CPUUtilization",
    Namespace: "AWS/EC2",
    Period: 60,
    Statistic: "Average",
    Threshold: 70.0,
    ActionsEnabled: false,
    AlarmDescription: "Alarm when server CPU exceeds 70%",
    Dimensions: [
      {
        Name: "InstanceId",
        Value: process.env.EC2_INSTANCE_ID, // Set the value of EC_INSTANCE_ID to
 the Id of an existing Amazon EC2 instance.
      },
    ],
    Unit: "Percent",
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
```

```
  }
};

export default run();
```

Create the client in a separate module and export it.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- For more information, see Amazon SDK for JavaScript Developer Guide.
- For API details, see PutMetricAlarm in *Amazon SDK for JavaScript API Reference*.

SDK for JavaScript (v2)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatch service object
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });

var params = {
  AlarmName: "Web_Server_CPU_Utilization",
  ComparisonOperator: "GreaterThanThreshold",
  EvaluationPeriods: 1,
  MetricName: "CPUUtilization",
  Namespace: "AWS/EC2",
  Period: 60,
  Statistic: "Average",
  Threshold: 70.0,
  ActionsEnabled: false,
```

```
  AlarmDescription: "Alarm when server CPU exceeds 70%",
  Dimensions: [
    {
      Name: "InstanceId",
      Value: "INSTANCE_ID",
    },
  ],
  Unit: "Percent",
};

cw.putMetricAlarm(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- For more information, see Amazon SDK for JavaScript Developer Guide.
- For API details, see PutMetricAlarm in *Amazon SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

```
suspend fun putMetricAlarm(
    alarmNameVal: String,
    instanceIdVal: String,
) {
    val dimensionOb =
        Dimension {
            name = "InstanceId"
            value = instanceIdVal
        }
```

```
    val request =
        PutMetricAlarmRequest {
            alarmName = alarmNameVal
            comparisonOperator = ComparisonOperator.GreaterThanThreshold
            evaluationPeriods = 1
            metricName = "CPUUtilization"
            namespace = "AWS/EC2"
            period = 60
            statistic = Statistic.fromValue("Average")
            threshold = 70.0
            actionsEnabled = false
            alarmDescription = "An Alarm created by the Kotlin SDK when server
 CPU utilization exceeds 70%"
            unit = StandardUnit.fromValue("Seconds")
            dimensions = listOf(dimensionOb)
        }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient.putMetricAlarm(request)
        println("Successfully created an alarm with name $alarmNameVal")
    }
}
```

- For API details, see PutMetricAlarm in *Amazon SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

```
class CloudWatchWrapper:
    """Encapsulates Amazon CloudWatch functions."""

    def __init__(self, cloudwatch_resource):
        """
```

```
        :param cloudwatch_resource: A Boto3 CloudWatch resource.
        """
        self.cloudwatch_resource = cloudwatch_resource


    def create_metric_alarm(
        self,
        metric_namespace,
        metric_name,
        alarm_name,
        stat_type,
        period,
        eval_periods,
        threshold,
        comparison_op,
    ):
        """
        Creates an alarm that watches a metric.

        :param metric_namespace: The namespace of the metric.
        :param metric_name: The name of the metric.
        :param alarm_name: The name of the alarm.
        :param stat_type: The type of statistic the alarm watches.
        :param period: The period in which metric data are grouped to calculate
                       statistics.
        :param eval_periods: The number of periods that the metric must be over
 the
                             alarm threshold before the alarm is set into an
 alarmed
                             state.
        :param threshold: The threshold value to compare against the metric
 statistic.
        :param comparison_op: The comparison operation used to compare the
 threshold
                              against the metric.
        :return: The newly created alarm.
        """
        try:
            metric = self.cloudwatch_resource.Metric(metric_namespace,
 metric_name)
            alarm = metric.put_alarm(
                AlarmName=alarm_name,
                Statistic=stat_type,
                Period=period,
```

```
                EvaluationPeriods=eval_periods,
                Threshold=threshold,
                ComparisonOperator=comparison_op,
            )
            logger.info(
                "Added alarm %s to track metric %s.%s.",
                alarm_name,
                metric_namespace,
                metric_name,
            )
        except ClientError:
            logger.exception(
                "Couldn't add alarm %s to metric %s.%s",
                alarm_name,
                metric_namespace,
                metric_name,
            )
            raise
        else:
            return alarm
```

- For API details, see PutMetricAlarm in *Amazon SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

```
# Creates or updates an alarm in Amazon CloudWatch.
#
# @param cloudwatch_client [Aws::CloudWatch::Client]
#   An initialized CloudWatch client.
# @param alarm_name [String] The name of the alarm.
# @param alarm_description [String] A description about the alarm.
# @param metric_name [String] The name of the metric associated with the alarm.
```

```
# @param alarm_actions [Array] A list of Strings representing the
#   Amazon Resource Names (ARNs) to execute when the alarm transitions to the
#   ALARM state.
# @param namespace [String] The namespace for the metric to alarm on.
# @param statistic [String] The statistic for the metric.
# @param dimensions [Array] A list of dimensions for the metric, specified as
#   Aws::CloudWatch::Types::Dimension.
# @param period [Integer] The number of seconds before re-evaluating the metric.
# @param unit [String] The unit of measure for the statistic.
# @param evaluation_periods [Integer] The number of periods over which data is
#   compared to the specified threshold.
# @param theshold [Float] The value against which the specified statistic is
 compared.
# @param comparison_operator [String] The arithmetic operation to use when
#   comparing the specified statistic and threshold.
# @return [Boolean] true if the alarm was created or updated; otherwise, false.
# @example
#   exit 1 unless alarm_created_or_updated?(
#     Aws::CloudWatch::Client.new(region: 'us-east-1'),
#     'ObjectsInBucket',
#     'Objects exist in this bucket for more than 1 day.',
#     'NumberOfObjects',
#     ['arn:aws:sns:us-east-1:111111111111:Default_CloudWatch_Alarms_Topic'],
#     'AWS/S3',
#     'Average',
#     [
#       {
#         name: 'BucketName',
#         value: 'amzn-s3-demo-bucket'
#       },
#       {
#         name: 'StorageType',
#         value: 'AllStorageTypes'
#       }
#     ],
#     86_400,
#     'Count',
#     1,
#     1,
#     'GreaterThanThreshold'
#   )
def alarm_created_or_updated?(
  cloudwatch_client,
  alarm_name,
```

```
  alarm_description,
  metric_name,
  alarm_actions,
  namespace,
  statistic,
  dimensions,
  period,
  unit,
  evaluation_periods,
  threshold,
  comparison_operator
)
  cloudwatch_client.put_metric_alarm(
    alarm_name: alarm_name,
    alarm_description: alarm_description,
    metric_name: metric_name,
    alarm_actions: alarm_actions,
    namespace: namespace,
    statistic: statistic,
    dimensions: dimensions,
    period: period,
    unit: unit,
    evaluation_periods: evaluation_periods,
    threshold: threshold,
    comparison_operator: comparison_operator
  )
  true
rescue StandardError => e
  puts "Error creating alarm: #{e.message}"
  false
end
```

- For API details, see PutMetricAlarm in *Amazon SDK for Ruby API Reference*.

SAP ABAP

SDK for SAP ABAP

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

```
TRY.
    lo_cwt->putmetricalarm(
      iv_alarmname                  = iv_alarm_name
      iv_comparisonoperator         = iv_comparison_operator
      iv_evaluationperiods          = iv_evaluation_periods
      iv_metricname                 = iv_metric_name
      iv_namespace                  = iv_namespace
      iv_statistic                  = iv_statistic
      iv_threshold                  = iv_threshold
      iv_actionsenabled             = iv_actions_enabled
      iv_alarmdescription           = iv_alarm_description
      iv_unit                       = iv_unit
      iv_period                     = iv_period
      it_dimensions                 = it_dimensions ).
    MESSAGE 'Alarm created.' TYPE 'I'.
  CATCH /aws1/cx_cwtlimitexceededfault.
    MESSAGE 'The request processing has exceeded the limit' TYPE 'E'.
ENDTRY.
```

- For API details, see PutMetricAlarm in *Amazon SDK for SAP ABAP API reference*.

For a complete list of Amazon SDK developer guides and code examples, see Using CloudWatch with an Amazon SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `PutMetricData` with an Amazon SDK or CLI

The following code examples show how to use `PutMetricData`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Learn the basics
- Manage metrics and alarms

.NET

Amazon SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

```
    /// <summary>
    /// Add some metric data using a call to a wrapper class.
    /// </summary>
    /// <param name="customMetricName">The metric name.</param>
    /// <param name="customMetricNamespace">The metric namespace.</param>
    /// <returns></returns>
    private static async Task<List<MetricDatum>> PutRandomMetricData(string 
 customMetricName,
        string customMetricNamespace)
    {
        List<MetricDatum> customData = new List<MetricDatum>();
        Random rnd = new Random();

        // Add 10 random values up to 100, starting with a timestamp 15 minutes 
 in the past.
        var utcNowMinus15 = DateTime.UtcNow.AddMinutes(-15);
        for (int i = 0; i < 10; i++)
        {
            var metricValue = rnd.Next(0, 100);
            customData.Add(
                new MetricDatum
                {
                    MetricName = customMetricName,
                    Value = metricValue,
```

```
                    TimestampUtc = utcNowMinus15.AddMinutes(i)
                }
            );
        }

        await _cloudWatchWrapper.PutMetricData(customMetricNamespace,
customData);
        return customData;
    }

    /// <summary>
    /// Wrapper to add metric data to a CloudWatch metric.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metricData">A data object for the metric data.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> PutMetricData(string metricNamespace,
        List<MetricDatum> metricData)
    {
        var putDataResponse = await _amazonCloudWatch.PutMetricDataAsync(
            new PutMetricDataRequest()
            {
                MetricData = metricData,
                Namespace = metricNamespace,
            });

        return putDataResponse.HttpStatusCode == HttpStatusCode.OK;
    }
```

- For API details, see PutMetricData in *Amazon SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

Include the required files.

```
#include <aws/core/Aws.h>
#include <aws/monitoring/CloudWatchClient.h>
#include <aws/monitoring/model/PutMetricDataRequest.h>
#include <iostream>
```

Put data into the metric.

```
        Aws::CloudWatch::CloudWatchClient cw;

        Aws::CloudWatch::Model::Dimension dimension;
        dimension.SetName("UNIQUE_PAGES");
        dimension.SetValue("URLS");

        Aws::CloudWatch::Model::MetricDatum datum;
        datum.SetMetricName("PAGES_VISITED");
        datum.SetUnit(Aws::CloudWatch::Model::StandardUnit::None);
        datum.SetValue(data_point);
        datum.AddDimensions(dimension);

        Aws::CloudWatch::Model::PutMetricDataRequest request;
        request.SetNamespace("SITE/TRAFFIC");
        request.AddMetricData(datum);

        auto outcome = cw.PutMetricData(request);
        if (!outcome.IsSuccess())
        {
            std::cout << "Failed to put sample metric data:" <<
                outcome.GetError().GetMessage() << std::endl;
        }
        else
        {
            std::cout << "Successfully put sample metric data" << std::endl;
        }
```

- For API details, see PutMetricData in *Amazon SDK for C++ API Reference*.

CLI

Amazon CLI

To publish a custom metric to Amazon CloudWatch

The following example uses the `put-metric-data` command to publish a custom metric to Amazon CloudWatch:

```
aws cloudwatch put-metric-data --namespace "Usage Metrics" --metric-data file://
metric.json
```

The values for the metric itself are stored in the JSON file, `metric.json`.

Here are the contents of that file:

```
[
  {
    "MetricName": "New Posts",
    "Timestamp": "Wednesday, June 12, 2013 8:28:20 PM",
    "Value": 0.50,
    "Unit": "Count"
  }
]
```

For more information, see Publishing Custom Metrics in the *Amazon CloudWatch Developer Guide*.

To specify multiple dimensions

The following example illustrates how to specify multiple dimensions. Each dimension is specified as a Name=Value pair. Multiple dimensions are separated by a comma.:

```
aws cloudwatch put-metric-data --metric-name Buffers --
namespace MyNameSpace --unit Bytes --value 231434333 --
dimensions InstanceID=1-23456789,InstanceType=m1.small
```

- For API details, see [PutMetricData](#) in *Amazon CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [Amazon Code Examples Repository](https://github.com/awsdocs/aws-doc-sdk-examples/tree/main/javav2/example_code/cloudwatch#code-examples).

```
    /**
     * Adds metric data for an alarm asynchronously.
     *
     * @param fileName the name of the JSON file containing the metric data
     * @return a CompletableFuture that asynchronously returns the
 PutMetricDataResponse
     */
    public CompletableFuture<PutMetricDataResponse>
 addMetricDataForAlarmAsync(String fileName) {
        CompletableFuture<String> readFileFuture =
 CompletableFuture.supplyAsync(() -> {
            try {
                JsonParser parser = new JsonFactory().createParser(new
 File(fileName));
                com.fasterxml.jackson.databind.JsonNode rootNode = new
 ObjectMapper().readTree(parser);
                return rootNode.toString(); // Return JSON as a string for
 further processing
            } catch (IOException e) {
                throw new RuntimeException("Failed to read file", e);
            }
        });

        return readFileFuture.thenCompose(jsonContent -> {
            try {
                com.fasterxml.jackson.databind.JsonNode rootNode = new
 ObjectMapper().readTree(jsonContent);
                String customMetricNamespace =
 rootNode.findValue("customMetricNamespace").asText();
                String customMetricName =
 rootNode.findValue("customMetricName").asText();
```

```
                Instant instant = Instant.now();

                // Create MetricDatum objects.
                MetricDatum datum1 = MetricDatum.builder()
                    .metricName(customMetricName)
                    .unit(StandardUnit.NONE)
                    .value(1001.00)
                    .timestamp(instant)
                    .build();

                MetricDatum datum2 = MetricDatum.builder()
                    .metricName(customMetricName)
                    .unit(StandardUnit.NONE)
                    .value(1002.00)
                    .timestamp(instant)
                    .build();

                List<MetricDatum> metricDataList = new ArrayList<>();
                metricDataList.add(datum1);
                metricDataList.add(datum2);

                // Build the PutMetricData request.
                PutMetricDataRequest request = PutMetricDataRequest.builder()
                    .namespace(customMetricNamespace)
                    .metricData(metricDataList)
                    .build();

                // Send the request asynchronously.
                return getAsyncClient().putMetricData(request);

            } catch (IOException e) {
                CompletableFuture<PutMetricDataResponse> failedFuture = new 
CompletableFuture<>();
                failedFuture.completeExceptionally(new RuntimeException("Failed 
to parse JSON content", e));
                return failedFuture;
            }
        }).whenComplete((response, exception) -> {
            if (exception != null) {
                logger.error("Failed to put metric data: " + 
exception.getMessage(), exception);
            } else {
                logger.info("Added metric values for metric.");
            }
```

```
        });
    }
```

- For API details, see PutMetricData in *Amazon SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

Import the SDK and client modules and call the API.

```
import { PutMetricDataCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

const run = async () => {
  // See https://docs.aws.amazon.com/AmazonCloudWatch/latest/APIReference/
API_PutMetricData.html#API_PutMetricData_RequestParameters
  // and https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/
publishingMetrics.html
  // for more information about the parameters in this command.
  const command = new PutMetricDataCommand({
    MetricData: [
      {
        MetricName: "PAGES_VISITED",
        Dimensions: [
          {
            Name: "UNIQUE_PAGES",
            Value: "URLS",
          },
        ],
        Unit: "None",
        Value: 1.0,
      },
    ],
    Namespace: "SITE/TRAFFIC",
```

```
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

Create the client in a separate module and export it.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- For more information, see Amazon SDK for JavaScript Developer Guide.
- For API details, see PutMetricData in *Amazon SDK for JavaScript API Reference*.

SDK for JavaScript (v2)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatch service object
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });

// Create parameters JSON for putMetricData
var params = {
  MetricData: [
```

```
    {
      MetricName: "PAGES_VISITED",
      Dimensions: [
        {
          Name: "UNIQUE_PAGES",
          Value: "URLS",
        },
      ],
      Unit: "None",
      Value: 1.0,
    },
  ],
  Namespace: "SITE/TRAFFIC",
};

cw.putMetricData(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", JSON.stringify(data));
  }
});
```

- For more information, see Amazon SDK for JavaScript Developer Guide.
- For API details, see PutMetricData in *Amazon SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

```
suspend fun addMetricDataForAlarm(fileName: String?) {
    // Read values from the JSON file.
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
```

```
    val customMetricNamespace =
 rootNode.findValue("customMetricNamespace").asText()
    val customMetricName = rootNode.findValue("customMetricName").asText()

    // Set an Instant object.
    val time =
 ZonedDateTime.now(ZoneOffset.UTC).format(DateTimeFormatter.ISO_INSTANT)
    val instant = Instant.parse(time)
    val datum =
        MetricDatum {
            metricName = customMetricName
            unit = StandardUnit.None
            value = 1001.00
            timestamp =
                aws.smithy.kotlin.runtime.time
                    .Instant(instant)
        }

    val datum2 =
        MetricDatum {
            metricName = customMetricName
            unit = StandardUnit.None
            value = 1002.00
            timestamp =
                aws.smithy.kotlin.runtime.time
                    .Instant(instant)
        }

    val metricDataList = ArrayList<MetricDatum>()
    metricDataList.add(datum)
    metricDataList.add(datum2)

    val request =
        PutMetricDataRequest {
            namespace = customMetricNamespace
            metricData = metricDataList
        }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient.putMetricData(request)
        println("Added metric values for for metric $customMetricName")
    }
}
```

- For API details, see PutMetricData in *Amazon SDK for Kotlin API reference*.

PowerShell

Tools for PowerShell

Example 1: Creates a new MetricDatum object, and writes it to Amazon Web Services CloudWatch Metrics.

```
### Create a MetricDatum .NET object
$Metric = New-Object -TypeName Amazon.CloudWatch.Model.MetricDatum
$Metric.Timestamp = [DateTime]::UtcNow
$Metric.MetricName = 'CPU'
$Metric.Value = 50

### Write the metric data to the CloudWatch service
Write-CWMetricData -Namespace instance1 -MetricData $Metric
```

- For API details, see PutMetricData in *Amazon Tools for PowerShell Cmdlet Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

```
class CloudWatchWrapper:
    """Encapsulates Amazon CloudWatch functions."""

    def __init__(self, cloudwatch_resource):
        """
        :param cloudwatch_resource: A Boto3 CloudWatch resource.
        """
        self.cloudwatch_resource = cloudwatch_resource
```

```
    def put_metric_data(self, namespace, name, value, unit):
        """
        Sends a single data value to CloudWatch for a metric. This metric is
 given
        a timestamp of the current UTC time.

        :param namespace: The namespace of the metric.
        :param name: The name of the metric.
        :param value: The value of the metric.
        :param unit: The unit of the metric.
        """
        try:
            metric = self.cloudwatch_resource.Metric(namespace, name)
            metric.put_data(
                Namespace=namespace,
                MetricData=[{"MetricName": name, "Value": value, "Unit": unit}],
            )
            logger.info("Put data for metric %s.%s", namespace, name)
        except ClientError:
            logger.exception("Couldn't put data for metric %s.%s", namespace,
 name)
            raise
```

Put a set of data into a CloudWatch metric.

```
class CloudWatchWrapper:
    """Encapsulates Amazon CloudWatch functions."""

    def __init__(self, cloudwatch_resource):
        """
        :param cloudwatch_resource: A Boto3 CloudWatch resource.
        """
        self.cloudwatch_resource = cloudwatch_resource


    def put_metric_data_set(self, namespace, name, timestamp, unit, data_set):
        """
        Sends a set of data to CloudWatch for a metric. All of the data in the
 set
```

```
        have the same timestamp and unit.

        :param namespace: The namespace of the metric.
        :param name: The name of the metric.
        :param timestamp: The UTC timestamp for the metric.
        :param unit: The unit of the metric.
        :param data_set: The set of data to send. This set is a dictionary that
                         contains a list of values and a list of corresponding
 counts.
                         The value and count lists must be the same length.
        """
        try:
            metric = self.cloudwatch_resource.Metric(namespace, name)
            metric.put_data(
                Namespace=namespace,
                MetricData=[
                    {
                        "MetricName": name,
                        "Timestamp": timestamp,
                        "Values": data_set["values"],
                        "Counts": data_set["counts"],
                        "Unit": unit,
                    }
                ],
            )
            logger.info("Put data set for metric %s.%s.", namespace, name)
        except ClientError:
            logger.exception("Couldn't put data set for metric %s.%s.",
 namespace, name)
            raise
```

- For API details, see [PutMetricData](#) in *Amazon SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [Amazon Code Examples Repository](https://github.com/awsdocs/aws-doc-sdk-examples).

```
require 'aws-sdk-cloudwatch'

# Adds a datapoint to a metric in Amazon CloudWatch.
#
# @param cloudwatch_client [Aws::CloudWatch::Client]
#   An initialized CloudWatch client.
# @param metric_namespace [String] The namespace of the metric to add the
#   datapoint to.
# @param metric_name [String] The name of the metric to add the datapoint to.
# @param dimension_name [String] The name of the dimension to add the
#   datapoint to.
# @param dimension_value [String] The value of the dimension to add the
#   datapoint to.
# @param metric_value [Float] The value of the datapoint.
# @param metric_unit [String] The unit of measurement for the datapoint.
# @return [Boolean]
# @example
#   exit 1 unless datapoint_added_to_metric?(
#     Aws::CloudWatch::Client.new(region: 'us-east-1'),
#     'SITE/TRAFFIC',
#     'UniqueVisitors',
#     'SiteName',
#     'example.com',
#     5_885.0,
#     'Count'
#   )
def datapoint_added_to_metric?(
  cloudwatch_client,
  metric_namespace,
  metric_name,
  dimension_name,
  dimension_value,
```

```
    metric_value,
    metric_unit
  )
    cloudwatch_client.put_metric_data(
      namespace: metric_namespace,
      metric_data: [
        {
          metric_name: metric_name,
          dimensions: [
            {
              name: dimension_name,
              value: dimension_value
            }
          ],
          value: metric_value,
          unit: metric_unit
        }
      ]
    )
    puts "Added data about '#{metric_name}' to namespace " \
      "'#{metric_namespace}'."
    true
  rescue StandardError => e
    puts "Error adding data about '#{metric_name}' to namespace " \
      "'#{metric_namespace}': #{e.message}"
    false
  end
```

- For API details, see PutMetricData in *Amazon SDK for Ruby API Reference*.

For a complete list of Amazon SDK developer guides and code examples, see Using CloudWatch with an Amazon SDK. This topic also includes information about getting started and details about previous SDK versions.

Scenarios for CloudWatch using Amazon SDKs

The following code examples show you how to implement common scenarios in CloudWatch with Amazon SDKs. These scenarios show you how to accomplish specific tasks by calling multiple functions within CloudWatch or combined with other Amazon Web Services services. Each scenario includes a link to the complete source code, where you can find instructions on how to set up and run the code.

Scenarios target an intermediate level of experience to help you understand service actions in context.

Examples

- Get started with CloudWatch alarms using an Amazon SDK
- Manage CloudWatch metrics and alarms using an Amazon SDK
- Monitor performance of Amazon DynamoDB using an Amazon SDK

Get started with CloudWatch alarms using an Amazon SDK

The following code example shows how to:

- Create an alarm.
- Disable alarm actions.
- Describe an alarm.
- Delete an alarm.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

```
DATA lt_alarmnames TYPE /aws1/cl_cwtalarmnames_w=>tt_alarmnames.
DATA lo_alarmname TYPE REF TO /aws1/cl_cwtalarmnames_w.

"Create an alarm"
TRY.
    lo_cwt->putmetricalarm(
      iv_alarmname                  = iv_alarm_name
```

```
          iv_comparisonoperator       = iv_comparison_operator
          iv_evaluationperiods        = iv_evaluation_periods
          iv_metricname               = iv_metric_name
          iv_namespace                = iv_namespace
          iv_statistic                = iv_statistic
          iv_threshold                = iv_threshold
          iv_actionsenabled           = iv_actions_enabled
          iv_alarmdescription         = iv_alarm_description
          iv_unit                     = iv_unit
          iv_period                   = iv_period
          it_dimensions               = it_dimensions ).
        MESSAGE 'Alarm created' TYPE 'I'.
      CATCH /aws1/cx_cwtlimitexceededfault.
        MESSAGE 'The request processing has exceeded the limit' TYPE 'E'.
    ENDTRY.

    "Create an ABAP internal table for the created alarm."
    lo_alarmname = NEW #( iv_value = iv_alarm_name ).
    INSERT lo_alarmname INTO TABLE lt_alarmnames.

    "Disable alarm actions."
    TRY.
        lo_cwt->disablealarmactions(
          it_alarmnames               = lt_alarmnames ).
        MESSAGE 'Alarm actions disabled' TYPE 'I'.
      CATCH /aws1/cx_rt_service_generic INTO DATA(lo_disablealarm_exception).
        DATA(lv_disablealarm_error) = |"{ lo_disablealarm_exception-
>av_err_code }" - { lo_disablealarm_exception->av_err_msg }|.
        MESSAGE lv_disablealarm_error TYPE 'E'.
    ENDTRY.

    "Describe alarm using the same ABAP internal table."
    TRY.
        oo_result = lo_cwt->describealarms(                       " oo_result is
 returned for testing purpose "
          it_alarmnames               = lt_alarmnames ).
        MESSAGE 'Alarms retrieved' TYPE 'I'.
      CATCH /aws1/cx_rt_service_generic INTO DATA(lo_describealarms_exception).
        DATA(lv_describealarms_error) = |"{ lo_describealarms_exception-
>av_err_code }" - { lo_describealarms_exception->av_err_msg }|.
        MESSAGE lv_describealarms_error TYPE 'E'.
    ENDTRY.

    "Delete alarm."
```

```
    TRY.
        lo_cwt->deletealarms(
          it_alarmnames = lt_alarmnames ).
        MESSAGE 'Alarms deleted' TYPE 'I'.
      CATCH /aws1/cx_cwtresourcenotfound.
        MESSAGE 'Resource being access is not found.' TYPE 'E'.
    ENDTRY.
```

- For API details, see the following topics in *Amazon SDK for SAP ABAP API reference*.
  - DeleteAlarms
  - DescribeAlarms
  - DisableAlarmActions
  - PutMetricAlarm

For a complete list of Amazon SDK developer guides and code examples, see Using CloudWatch with an Amazon SDK. This topic also includes information about getting started and details about previous SDK versions.

Manage CloudWatch metrics and alarms using an Amazon SDK

The following code example shows how to:

- Create an alarm to watch a CloudWatch metric.
- Put data into a metric and trigger the alarm.
- Get data from the alarm.
- Delete the alarm.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Amazon Code Examples Repository.

Create a class that wraps CloudWatch operations.

```
from datetime import datetime, timedelta
import logging
from pprint import pprint
import random
import time
import boto3
from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)


class CloudWatchWrapper:
    """Encapsulates Amazon CloudWatch functions."""

    def __init__(self, cloudwatch_resource):
        """
        :param cloudwatch_resource: A Boto3 CloudWatch resource.
        """
        self.cloudwatch_resource = cloudwatch_resource


    def put_metric_data_set(self, namespace, name, timestamp, unit, data_set):
        """
        Sends a set of data to CloudWatch for a metric. All of the data in the
 set
        have the same timestamp and unit.

        :param namespace: The namespace of the metric.
        :param name: The name of the metric.
        :param timestamp: The UTC timestamp for the metric.
        :param unit: The unit of the metric.
        :param data_set: The set of data to send. This set is a dictionary that
                         contains a list of values and a list of corresponding
 counts.
                         The value and count lists must be the same length.
        """
        try:
            metric = self.cloudwatch_resource.Metric(namespace, name)
            metric.put_data(
                Namespace=namespace,
                MetricData=[
                    {
```

```
                        "MetricName": name,
                        "Timestamp": timestamp,
                        "Values": data_set["values"],
                        "Counts": data_set["counts"],
                        "Unit": unit,
                    }
                ],
            )
            logger.info("Put data set for metric %s.%s.", namespace, name)
        except ClientError:
            logger.exception("Couldn't put data set for metric %s.%s.",
namespace, name)
            raise


    def create_metric_alarm(
        self,
        metric_namespace,
        metric_name,
        alarm_name,
        stat_type,
        period,
        eval_periods,
        threshold,
        comparison_op,
    ):
        """
        Creates an alarm that watches a metric.

        :param metric_namespace: The namespace of the metric.
        :param metric_name: The name of the metric.
        :param alarm_name: The name of the alarm.
        :param stat_type: The type of statistic the alarm watches.
        :param period: The period in which metric data are grouped to calculate
                       statistics.
        :param eval_periods: The number of periods that the metric must be over
the
                             alarm threshold before the alarm is set into an
alarmed
                             state.
        :param threshold: The threshold value to compare against the metric
statistic.
        :param comparison_op: The comparison operation used to compare the
threshold
```

```
                                against the metric.
        :return: The newly created alarm.
        """
        try:
            metric = self.cloudwatch_resource.Metric(metric_namespace,
metric_name)
            alarm = metric.put_alarm(
                AlarmName=alarm_name,
                Statistic=stat_type,
                Period=period,
                EvaluationPeriods=eval_periods,
                Threshold=threshold,
                ComparisonOperator=comparison_op,
            )
            logger.info(
                "Added alarm %s to track metric %s.%s.",
                alarm_name,
                metric_namespace,
                metric_name,
            )
        except ClientError:
            logger.exception(
                "Couldn't add alarm %s to metric %s.%s",
                alarm_name,
                metric_namespace,
                metric_name,
            )
            raise
        else:
            return alarm


    def put_metric_data(self, namespace, name, value, unit):
        """
        Sends a single data value to CloudWatch for a metric. This metric is
given
        a timestamp of the current UTC time.

        :param namespace: The namespace of the metric.
        :param name: The name of the metric.
        :param value: The value of the metric.
        :param unit: The unit of the metric.
        """
        try:
```

```
            metric = self.cloudwatch_resource.Metric(namespace, name)
            metric.put_data(
                Namespace=namespace,
                MetricData=[{"MetricName": name, "Value": value, "Unit": unit}],
            )
            logger.info("Put data for metric %s.%s", namespace, name)
        except ClientError:
            logger.exception("Couldn't put data for metric %s.%s", namespace,
name)
            raise


    def get_metric_statistics(self, namespace, name, start, end, period,
stat_types):
        """
        Gets statistics for a metric within a specified time span. Metrics are
grouped
        into the specified period.

        :param namespace: The namespace of the metric.
        :param name: The name of the metric.
        :param start: The UTC start time of the time span to retrieve.
        :param end: The UTC end time of the time span to retrieve.
        :param period: The period, in seconds, in which to group metrics. The
period
                       must match the granularity of the metric, which depends on
                       the metric's age. For example, metrics that are older than
                       three hours have a one-minute granularity, so the period
must
                       be at least 60 and must be a multiple of 60.
        :param stat_types: The type of statistics to retrieve, such as average
value
                           or maximum value.
        :return: The retrieved statistics for the metric.
        """
        try:
            metric = self.cloudwatch_resource.Metric(namespace, name)
            stats = metric.get_statistics(
                StartTime=start, EndTime=end, Period=period,
Statistics=stat_types
            )
            logger.info(
                "Got %s statistics for %s.", len(stats["Datapoints"]),
stats["Label"]
```

```
            )
        except ClientError:
            logger.exception("Couldn't get statistics for %s.%s.", namespace, 
name)
            raise
        else:
            return stats


    def get_metric_alarms(self, metric_namespace, metric_name):
        """
        Gets the alarms that are currently watching the specified metric.

        :param metric_namespace: The namespace of the metric.
        :param metric_name: The name of the metric.
        :returns: An iterator that yields the alarms.
        """
        metric = self.cloudwatch_resource.Metric(metric_namespace, metric_name)
        alarm_iter = metric.alarms.all()
        logger.info("Got alarms for metric %s.%s.", metric_namespace, 
metric_name)
        return alarm_iter


    def delete_metric_alarms(self, metric_namespace, metric_name):
        """
        Deletes all of the alarms that are currently watching the specified 
metric.

        :param metric_namespace: The namespace of the metric.
        :param metric_name: The name of the metric.
        """
        try:
            metric = self.cloudwatch_resource.Metric(metric_namespace, 
metric_name)
            metric.alarms.delete()
            logger.info(
                "Deleted alarms for metric %s.%s.", metric_namespace, metric_name
            )
        except ClientError:
            logger.exception(
                "Couldn't delete alarms for metric %s.%s.",
                metric_namespace,
                metric_name,
```

```
            )
            raise
```

Use the wrapper class to put data in a metric, trigger an alarm that watches the metric, and get data from the alarm.

```
def usage_demo():
    print("-" * 88)
    print("Welcome to the Amazon CloudWatch metrics and alarms demo!")
    print("-" * 88)

    logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")

    cw_wrapper = CloudWatchWrapper(boto3.resource("cloudwatch"))

    minutes = 20
    metric_namespace = "doc-example-metric"
    metric_name = "page_views"
    start = datetime.utcnow() - timedelta(minutes=minutes)
    print(
        f"Putting data into metric {metric_namespace}.{metric_name} spanning the
 "
        f"last {minutes} minutes."
    )
    for offset in range(0, minutes):
        stamp = start + timedelta(minutes=offset)
        cw_wrapper.put_metric_data_set(
            metric_namespace,
            metric_name,
            stamp,
            "Count",
            {
                "values": [
                    random.randint(bound, bound * 2)
                    for bound in range(offset + 1, offset + 11)
                ],
                "counts": [random.randint(1, offset + 1) for _ in range(10)],
            },
        )
```

```
    alarm_name = "high_page_views"
    period = 60
    eval_periods = 2
    print(f"Creating alarm {alarm_name} for metric {metric_name}.")
    alarm = cw_wrapper.create_metric_alarm(
        metric_namespace,
        metric_name,
        alarm_name,
        "Maximum",
        period,
        eval_periods,
        100,
        "GreaterThanThreshold",
    )
    print(f"Alarm ARN is {alarm.alarm_arn}.")
    print(f"Current alarm state is: {alarm.state_value}.")

    print(
        f"Sending data to trigger the alarm. This requires data over the
 threshold "
        f"for {eval_periods} periods of {period} seconds each."
    )
    while alarm.state_value == "INSUFFICIENT_DATA":
        print("Sending data for the metric.")
        cw_wrapper.put_metric_data(
            metric_namespace, metric_name, random.randint(100, 200), "Count"
        )
        alarm.load()
        print(f"Current alarm state is: {alarm.state_value}.")
        if alarm.state_value == "INSUFFICIENT_DATA":
            print(f"Waiting for {period} seconds...")
            time.sleep(period)
        else:
            print("Wait for a minute for eventual consistency of metric data.")
            time.sleep(period)
            if alarm.state_value == "OK":
                alarm.load()
                print(f"Current alarm state is: {alarm.state_value}.")

    print(
        f"Getting data for metric {metric_namespace}.{metric_name} during
 timespan "
        f"of {start} to {datetime.utcnow()} (times are UTC)."
    )
```

```
    stats = cw_wrapper.get_metric_statistics(
        metric_namespace,
        metric_name,
        start,
        datetime.utcnow(),
        60,
        ["Average", "Minimum", "Maximum"],
    )
    print(
        f"Got {len(stats['Datapoints'])} data points for metric "
        f"{metric_namespace}.{metric_name}."
    )
    pprint(sorted(stats["Datapoints"], key=lambda x: x["Timestamp"]))

    print(f"Getting alarms for metric {metric_name}.")
    alarms = cw_wrapper.get_metric_alarms(metric_namespace, metric_name)
    for alarm in alarms:
        print(f"Alarm {alarm.name} is currently in state {alarm.state_value}.")

    print(f"Deleting alarms for metric {metric_name}.")
    cw_wrapper.delete_metric_alarms(metric_namespace, metric_name)

    print("Thanks for watching!")
    print("-" * 88)
```

- For API details, see the following topics in *Amazon SDK for Python (Boto3) API Reference*.
  - DeleteAlarms
  - DescribeAlarmsForMetric
  - DisableAlarmActions
  - EnableAlarmActions
  - GetMetricStatistics
  - ListMetrics
  - PutMetricAlarm
  - PutMetricData

For a complete list of Amazon SDK developer guides and code examples, see Using CloudWatch with an Amazon SDK. This topic also includes information about getting started and details about previous SDK versions.

Monitor performance of Amazon DynamoDB using an Amazon SDK

The following code example shows how to configure an application's use of DynamoDB to monitor performance.

Java

SDK for Java 2.x

This example shows how to configure a Java application to monitor the performance of DynamoDB. The application sends metric data to CloudWatch where you can monitor the performance.

For complete source code and instructions on how to set up and run, see the full example on GitHub.

Services used in this example

- CloudWatch
- DynamoDB

For a complete list of Amazon SDK developer guides and code examples, see Using CloudWatch with an Amazon SDK. This topic also includes information about getting started and details about previous SDK versions.

Security in Amazon CloudWatch

Cloud security at Amazon is the highest priority. As an Amazon customer, you benefit from a data center and network architecture that is built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between Amazon and you. The shared responsibility model describes this as security of the cloud and security in the cloud:

- **Security of the cloud** – Amazon is responsible for protecting the infrastructure that runs Amazon services in the Amazon Cloud. Amazon also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the Amazon Compliance Programs. To learn about the compliance programs that apply to CloudWatch, see Amazon Services in Scope by Compliance Program.
- **Security in the cloud** – Your responsibility is determined by the Amazon service that you use. You are also responsible for other factors including the sensitivity of your data, your company's requirements, and applicable laws and regulations

This documentation helps you understand how to apply the shared responsibility model when using Amazon CloudWatch. It shows you how to configure Amazon CloudWatch to meet your security and compliance objectives. You also learn how to use other Amazon services that help you to monitor and secure your CloudWatch resources.

Contents

Data protection in Amazon CloudWatch

The Amazon shared responsibility model applies to data protection in Amazon CloudWatch. As described in this model, Amazon is responsible for protecting the global infrastructure that runs all of the Amazon Web Services Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. You are also responsible for the security configuration and management tasks for the Amazon Web Services services that you use. For more information about data privacy, see the Data Privacy FAQ.

For data protection purposes, we recommend that you protect Amazon Web Services account credentials and set up individual users with Amazon IAM Identity Center or Amazon Identity and Access Management (IAM). That way, each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with Amazon resources. We require TLS 1.2 and recommend TLS 1.3.
- Set up API and user activity logging with Amazon CloudTrail. For information about using CloudTrail trails to capture Amazon activities, see Working with CloudTrail trails in the *Amazon CloudTrail User Guide*.
- Use Amazon encryption solutions, along with all default security controls within Amazon Web Services services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing sensitive data that is stored in Amazon S3.
- If you require FIPS 140-3 validated cryptographic modules when accessing Amazon through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see Federal Information Processing Standard (FIPS) 140-3.

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form text fields such as a **Name** field. This includes when you work with CloudWatch or other Amazon Web Services services using the console, API, Amazon CLI, or Amazon SDKs. Any data that you enter into tags or free-form text fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

Encryption in transit

CloudWatch uses end-to-end encryption of data in transit. For more information, see Encryption of data in transit.

Identity and access management for Amazon CloudWatch

Amazon Identity and Access Management (IAM) is an Amazon Web Services service that helps an administrator securely control access to Amazon resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use CloudWatch resources. IAM is an Amazon Web Services service that you can use with no additional charge.

Topics

Audience

How you use Amazon Identity and Access Management (IAM) differs, depending on the work that you do in CloudWatch.

Service user – If you use the CloudWatch service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more CloudWatch features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator. If you cannot access a feature in CloudWatch, see Troubleshooting Amazon CloudWatch identity and access.

Service administrator – If you're in charge of CloudWatch resources at your company, you probably have full access to CloudWatch. It's your job to determine which CloudWatch features and resources your service users should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with CloudWatch, see How Amazon CloudWatch works with IAM.

IAM administrator – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to CloudWatch. To view example CloudWatch identity-based policies that you can use in IAM, see Identity-based policy examples for Amazon CloudWatch.

Authenticating with identities

Authentication is how you sign in to Amazon using your identity credentials. You must be *authenticated* (signed in to Amazon) as the Amazon Web Services account root user, as an IAM user, or by assuming an IAM role.

If you access Amazon programmatically, Amazon provides a software development kit (SDK) and a command line interface (CLI) to cryptographically sign your requests by using your credentials. If you don't use Amazon tools, you must sign requests yourself. For more information about using the recommended method to sign requests yourself, see Amazon Signature Version 4 for API requests in the *IAM User Guide*.

Regardless of the authentication method that you use, you might be required to provide additional security information. For example, Amazon recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more, see Amazon Multi-factor authentication in IAM in the *IAM User Guide*.

Amazon Web Services account root user

When you create an Amazon Web Services account, you begin with one sign-in identity that has complete access to all Amazon Web Services services and resources in the account. This identity is called the Amazon Web Services account *root user* and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you don't use the root user for your everyday tasks. Safeguard your root user credentials and use them to perform the tasks that only the root user can perform. For the complete list of tasks that require you to sign in as the root user, see Tasks that require root user credentials in the *IAM User Guide*.

Federated identity

As a best practice, require human users, including users that require administrator access, to use federation with an identity provider to access Amazon Web Services services by using temporary credentials.

A *federated identity* is a user from your enterprise user directory, a web identity provider, the Amazon Directory Service, or any user that accesses Amazon Web Services services by using credentials provided through an identity source. When federated identities access Amazon Web Services accounts, they assume roles, and the roles provide temporary credentials.

IAM users and groups

An *IAM user* is an identity within your Amazon Web Services account that has specific permissions for a single person or application. Where possible, we recommend relying on temporary credentials instead of creating IAM users who have long-term credentials such as passwords and access keys. However, if you have specific use cases that require long-term credentials with IAM users, we recommend that you rotate access keys. For more information, see Rotate access keys regularly for use cases that require long-term credentials in the *IAM User Guide*.

An *IAM group* is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named *IAMAdmins* and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see Use cases for IAM users in the *IAM User Guide*.

IAM roles

An *IAM role* is an identity within your Amazon Web Services account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. To temporarily assume an IAM role in the Amazon Web Services Management Console, you can switch from a user to an IAM role (console). You can assume a role by calling an Amazon CLI or Amazon API operation or by using a custom URL. For more information about methods for using roles, see Methods to assume a role in the *IAM User Guide*.

IAM roles with temporary credentials are useful in the following situations:

- **Federated user access** – To assign permissions to a federated identity, you create a role and define permissions for the role. When a federated identity authenticates, the identity is associated with the role and is granted the permissions that are defined by the role. For information about roles for federation, see Create a role for a third-party identity provider (federation) in the *IAM User Guide*.
- **Temporary IAM user permissions** – An IAM user or role can assume an IAM role to temporarily take on different permissions for a specific task.
- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some Amazon Web Services services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see Cross account resource access in IAM in the *IAM User Guide*.
- **Cross-service access** – Some Amazon Web Services services use features in other Amazon Web Services services. For example, when you make a call in a service, it's common for that service to run applications in Amazon EC2 or store objects in Amazon S3. A service might do this using the calling principal's permissions, using a service role, or using a service-linked role.
  - **Forward access sessions (FAS)** – When you use an IAM user or role to perform actions in Amazon, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an Amazon Web Services service, combined with the requesting Amazon Web Services service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other Amazon Web Services services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see Forward access sessions.

  - **Service role –** A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Create a role to delegate permissions to an Amazon Web Services service in the *IAM User Guide*.
  - **Service-linked role –** A service-linked role is a type of service role that is linked to an Amazon Web Services service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your Amazon Web Services account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.
- **Applications running on Amazon EC2 –** You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making Amazon CLI or Amazon API requests. This is preferable to storing access keys within the EC2 instance. To assign an Amazon role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see Use an IAM role to grant permissions to applications running on Amazon EC2 instances in the *IAM User Guide*.

Managing access using policies

You control access in Amazon by creating policies and attaching them to Amazon identities or resources. A policy is an object in Amazon that, when associated with an identity or resource, defines their permissions. Amazon evaluates these policies when a principal (user, root user, or role session) makes a request. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in Amazon as JSON documents. For more information about the structure and contents of JSON policy documents, see Overview of JSON policies in the *IAM User Guide*.

Administrators can use Amazon JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

By default, users and roles have no permissions. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action.

A user with that policy can get role information from the Amazon Web Services Management Console, the Amazon CLI, or the Amazon API.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Define custom IAM permissions with customer managed policies in the *IAM User Guide*.

Identity-based policies can be further categorized as *inline policies* or *managed policies*. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that you can attach to multiple users, groups, and roles in your Amazon Web Services account. Managed policies include Amazon managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see Choose between managed policies and inline policies in the *IAM User Guide*.

Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or Amazon Web Services services.

Resource-based policies are inline policies that are located in that service. You can't use Amazon managed policies from IAM in a resource-based policy.

Access control lists (ACLs)

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Amazon S3, Amazon WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see Access control list (ACL) overview in the *Amazon Simple Storage Service Developer Guide*.

Other policy types

Amazon supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries –** A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of an entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the `Principal` field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see Permissions boundaries for IAM entities in the *IAM User Guide*.
- **Service control policies (SCPs) –** SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in Amazon Organizations. Amazon Organizations is a service for grouping and centrally managing multiple Amazon Web Services accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each Amazon Web Services account root user. For more information about Organizations and SCPs, see Service control policies in the *Amazon Organizations User Guide*.
- **Resource control policies (RCPs) –** RCPs are JSON policies that you can use to set the maximum available permissions for resources in your accounts without updating the IAM policies attached to each resource that you own. The RCP limits permissions for resources in member accounts and can impact the effective permissions for identities, including the Amazon Web Services account root user, regardless of whether they belong to your organization. For more information about Organizations and RCPs, including a list of Amazon Web Services services that support RCPs, see Resource control policies (RCPs) in the *Amazon Organizations User Guide*.
- **Session policies –** Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's permissions are the intersection of the user or role's identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. For more information, see Session policies in the *IAM User Guide*.

Multiple policy types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how Amazon determines whether to allow a request when multiple policy types are involved, see Policy evaluation logic in the *IAM User Guide*.

How Amazon CloudWatch works with IAM

Before you use IAM to manage access to CloudWatch, learn what IAM features are available to use with CloudWatch.

The following tables list the IAM features that you can use with Amazon CloudWatch.

IAM feature	CloudWatch support
Identity-based policies	Yes
Resource-based policies	No
Policy actions	Yes
Policy resources	Yes
Policy condition keys (service-specific)	Yes
ACLs	No
ABAC (tags in policies)	Partial
Temporary credentials	Yes
Principal permissions	Yes
Service roles	Yes
Service-linked roles	No

To get a high-level view of how CloudWatch and other Amazon services work with most IAM features, see Amazon services that work with IAM in the *IAM User Guide*.

Identity-based policies for CloudWatch

Supports identity-based policies: Yes

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Define custom IAM permissions with customer managed policies in the *IAM User Guide*.

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. You can't specify the principal in an identity-based policy because it applies to the user or role to which it is attached. To learn about all of the elements that you can use in a JSON policy, see IAM JSON policy elements reference in the *IAM User Guide*.

Identity-based policy examples for CloudWatch

To view examples of CloudWatch identity-based policies, see Identity-based policy examples for Amazon CloudWatch.

Resource-based policies within CloudWatch

Supports resource-based policies: No

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or Amazon Web Services services.

To enable cross-account access, you can specify an entire account or IAM entities in another account as the principal in a resource-based policy. Adding a cross-account principal to a resource-based policy is only half of establishing the trust relationship. When the principal and the resource are in different Amazon Web Services accounts, an IAM administrator in the trusted account must also grant the principal entity (user or role) permission to access the resource. They grant permission by attaching an identity-based policy to the entity. However, if a resource-based policy grants access to a principal in the same account, no additional identity-based policy is required. For more information, see Cross account resource access in IAM in the *IAM User Guide*.

Policy actions for CloudWatch

Supports policy actions: Yes

Administrators can use Amazon JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Action` element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Policy actions usually have the same name as the associated Amazon API operation. There are some exceptions, such as *permission-only actions* that don't have a matching API operation. There are also some operations that require multiple actions in a policy. These additional actions are called *dependent actions*.

Include actions in a policy to grant permissions to perform the associated operation.

To see a list of CloudWatch actions, see Actions defined by Amazon CloudWatch in the *Service Authorization Reference*.

Policy actions in CloudWatch use the following prefix before the action:

```
cloudwatch
```

To specify multiple actions in a single statement, separate them with commas.

```
"Action": [
      "cloudwatch:action1",
      "cloudwatch:action2"
         ]
```

To view examples of CloudWatch identity-based policies, see Identity-based policy examples for Amazon CloudWatch.

Policy resources for CloudWatch

Supports policy resources: Yes

Administrators can use Amazon JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Resource` JSON policy element specifies the object or objects to which the action applies. Statements must include either a `Resource` or a `NotResource` element. As a best practice, specify a resource using its Amazon Resource Name (ARN). You can do this for actions that support a specific resource type, known as *resource-level permissions*.

For actions that don't support resource-level permissions, such as listing operations, use a wildcard (*) to indicate that the statement applies to all resources.

```
"Resource": "*"
```

To see a list of CloudWatch resource types and their ARNs, see Resources defined by Amazon CloudWatch in the *Service Authorization Reference*. To learn with which actions you can specify the ARN of each resource, see Actions defined by Amazon CloudWatch.

To view examples of CloudWatch identity-based policies, see Identity-based policy examples for Amazon CloudWatch.

Policy condition keys for CloudWatch

Supports service-specific policy condition keys: Yes

Administrators can use Amazon JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Condition` element (or `Condition` *block*) lets you specify conditions in which a statement is in effect. The `Condition` element is optional. You can create conditional expressions that use condition operators, such as equals or less than, to match the condition in the policy with values in the request.

If you specify multiple `Condition` elements in a statement, or multiple keys in a single `Condition` element, Amazon evaluates them using a logical `AND` operation. If you specify multiple values for a single condition key, Amazon evaluates the condition using a logical `OR` operation. All of the conditions must be met before the statement's permissions are granted.

You can also use placeholder variables when you specify conditions. For example, you can grant an IAM user permission to access a resource only if it is tagged with their IAM user name. For more information, see IAM policy elements: variables and tags in the *IAM User Guide*.

Amazon supports global condition keys and service-specific condition keys. To see all Amazon global condition keys, see Amazon global condition context keys in the *IAM User Guide*.

To see a list of CloudWatch condition keys, see Condition keys for Amazon CloudWatch in the *Service Authorization Reference*. To learn with which actions and resources you can use a condition key, see Actions defined by Amazon CloudWatch.

To view examples of CloudWatch identity-based policies, see Identity-based policy examples for Amazon CloudWatch.

ACLs in CloudWatch

Supports ACLs: No

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

ABAC with CloudWatch

Supports ABAC (tags in policies): Partial

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes. In Amazon, these attributes are called *tags*. You can attach tags to IAM entities (users or roles) and to many Amazon resources. Tagging entities and resources is the first step of ABAC. Then you design ABAC policies to allow operations when the principal's tag matches the tag on the resource that they are trying to access.

ABAC is helpful in environments that are growing rapidly and helps with situations where policy management becomes cumbersome.

To control access based on tags, you provide tag information in the condition element of a policy using the `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys.

If a service supports all three condition keys for every resource type, then the value is **Yes** for the service. If a service supports all three condition keys for only some resource types, then the value is **Partial**.

For more information about ABAC, see Define permissions with ABAC authorization in the *IAM User Guide*. To view a tutorial with steps for setting up ABAC, see Use attribute-based access control (ABAC) in the *IAM User Guide*.

Using temporary credentials with CloudWatch

Supports temporary credentials: Yes

Some Amazon Web Services services don't work when you sign in using temporary credentials. For additional information, including which Amazon Web Services services work with temporary credentials, see Amazon Web Services services that work with IAM in the *IAM User Guide*.

You are using temporary credentials if you sign in to the Amazon Web Services Management Console using any method except a user name and password. For example, when you access Amazon using your company's single sign-on (SSO) link, that process automatically creates temporary credentials. You also automatically create temporary credentials when you sign in to the console as a user and then switch roles. For more information about switching roles, see Switch from a user to an IAM role (console) in the *IAM User Guide*.

You can manually create temporary credentials using the Amazon CLI or Amazon API. You can then use those temporary credentials to access Amazon. Amazon recommends that you dynamically generate temporary credentials instead of using long-term access keys. For more information, see Temporary security credentials in IAM.

Cross-service principal permissions for CloudWatch

Supports forward access sessions (FAS): Yes

When you use an IAM user or role to perform actions in Amazon, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an Amazon Web Services service, combined with the requesting Amazon Web Services service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other Amazon Web Services services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see Forward access sessions.

Service roles for CloudWatch

Supports service roles: Yes

A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Create a role to delegate permissions to an Amazon Web Services service in the *IAM User Guide*.

> **Warning**
>
> Changing the permissions for a service role might break CloudWatch functionality. Edit service roles only when CloudWatch provides guidance to do so.

Identity-based policy examples for Amazon CloudWatch

By default, users and roles don't have permission to create or modify CloudWatch resources. They also can't perform tasks by using the Amazon Web Services Management Console, Amazon Command Line Interface (Amazon CLI), or Amazon API. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

To learn how to create an IAM identity-based policy by using these example JSON policy documents, see Create IAM policies (console) in the *IAM User Guide*.

For details about actions and resource types defined by CloudWatch, including the format of the ARNs for each of the resource types, see Actions, resources, and condition keys for Amazon CloudWatch in the *Service Authorization Reference*.

Topics

- Policy best practices
- Using the CloudWatch console

Policy best practices

Identity-based policies determine whether someone can create, access, or delete CloudWatch resources in your account. These actions can incur costs for your Amazon Web Services account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get started with Amazon managed policies and move toward least-privilege permissions** – To get started granting permissions to your users and workloads, use the *Amazon managed policies* that grant permissions for many common use cases. They are available in your Amazon Web Services account. We recommend that you reduce permissions further by defining Amazon customer managed policies that are specific to your use cases. For more information, see Amazon managed policies or Amazon managed policies for job functions in the *IAM User Guide*.

- **Apply least-privilege permissions** – When you set permissions with IAM policies, grant only the permissions required to perform a task. You do this by defining the actions that can be taken on specific resources under specific conditions, also known as *least-privilege permissions*. For more information about using IAM to apply permissions, see Policies and permissions in IAM in the *IAM User Guide*.
- **Use conditions in IAM policies to further restrict access** – You can add a condition to your policies to limit access to actions and resources. For example, you can write a policy condition to specify that all requests must be sent using SSL. You can also use conditions to grant access to service actions if they are used through a specific Amazon Web Services service, such as Amazon CloudFormation. For more information, see IAM JSON policy elements: Condition in the *IAM User Guide*.
- **Use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions** – IAM Access Analyzer validates new and existing policies so that the policies adhere to the IAM policy language (JSON) and IAM best practices. IAM Access Analyzer provides more than 100 policy checks and actionable recommendations to help you author secure and functional policies. For more information, see Validate policies with IAM Access Analyzer in the *IAM User Guide*.
- **Require multi-factor authentication (MFA)** – If you have a scenario that requires IAM users or a root user in your Amazon Web Services account, turn on MFA for additional security. To require MFA when API operations are called, add MFA conditions to your policies. For more information, see Secure API access with MFA in the *IAM User Guide*.

For more information about best practices in IAM, see Security best practices in IAM in the *IAM User Guide*.

Using the CloudWatch console

To access the Amazon CloudWatch console, you must have a minimum set of permissions. These permissions must allow you to list and view details about the CloudWatch resources in your Amazon Web Services account. If you create an identity-based policy that is more restrictive than the minimum required permissions, the console won't function as intended for entities (users or roles) with that policy.

You don't need to allow minimum console permissions for users that are making calls only to the Amazon CLI or the Amazon API. Instead, allow access to only the actions that match the API operation that they're trying to perform.

To ensure that users and roles can still use the CloudWatch console, also attach the CloudWatch `ConsoleAccess` or `ReadOnly` Amazon managed policy to the entities. For more information, see [Adding permissions to a user](#) in the *IAM User Guide*.

Permissions needed for CloudWatch console

The full set of permissions required to work with the CloudWatch console are listed below. These permissions provide full write and read access to the CloudWatch console.

- application-autoscaling:DescribeScalingPolicies
- autoscaling:DescribeAutoScalingGroups
- autoscaling:DescribePolicies
- cloudtrail:DescribeTrails
- cloudwatch:DeleteAlarms
- cloudwatch:DescribeAlarmHistory
- cloudwatch:DescribeAlarms
- cloudwatch:GetMetricData
- cloudwatch:GetMetricStatistics
- cloudwatch:ListMetrics
- cloudwatch:PutMetricAlarm
- cloudwatch:PutMetricData
- ec2:DescribeInstances
- ec2:DescribeTags
- ec2:DescribeVolumes
- es:DescribeElasticsearchDomain
- es:ListDomainNames
- events:DeleteRule
- events:DescribeRule
- events:DisableRule
- events:EnableRule
- events:ListRules
- events:PutRule

- iam:AttachRolePolicy
- iam:CreateRole
- iam:GetPolicy
- iam:GetPolicyVersion
- iam:GetRole
- iam:ListAttachedRolePolicies
- iam:ListRoles
- kinesis:DescribeStream
- kinesis:ListStreams
- lambda:AddPermission
- lambda:CreateFunction
- lambda:GetFunctionConfiguration
- lambda:ListAliases
- lambda:ListFunctions
- lambda:ListVersionsByFunction
- lambda:RemovePermission
- logs:CancelExportTask
- logs:CreateExportTask
- logs:CreateLogGroup
- logs:CreateLogStream
- logs:DeleteLogGroup
- logs:DeleteLogStream
- logs:DeleteMetricFilter
- logs:DeleteRetentionPolicy
- logs:DeleteSubscriptionFilter
- logs:DescribeExportTasks
- logs:DescribeLogGroups
- logs:DescribeLogStreams
- logs:DescribeMetricFilters
- logs:DescribeQueries

- logs:DescribeSubscriptionFilters
- logs:FilterLogEvents
- logs:GetLogGroupFields
- logs:GetLogRecord
- logs:GetLogEvents
- logs:GetQueryResults
- logs:PutMetricFilter
- logs:PutRetentionPolicy
- logs:PutSubscriptionFilter
- logs:StartQuery
- logs:StopQuery
- logs:TestMetricFilter
- s3:CreateBucket
- s3:ListBucket
- sns:CreateTopic
- sns:GetTopicAttributes
- sns:ListSubscriptions
- sns:ListTopics
- sns:SetTopicAttributes
- sns:Subscribe
- sns:Unsubscribe
- sqs:GetQueueAttributes
- sqs:GetQueueUrl
- sqs:ListQueues
- sqs:SetQueueAttributes
- swf:CreateAction
- swf:DescribeAction
- swf:ListActionTemplates
- swf:RegisterAction
- swf:RegisterDomain

- swf:UpdateAction

Additionally, to view the X-Ray Trace Map, you need `AWSXrayReadOnlyAccess`

Troubleshooting Amazon CloudWatch identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with CloudWatch and IAM.

Topics

I am not authorized to perform an action in CloudWatch

If you receive an error that you're not authorized to perform an action, your policies must be updated to allow you to perform the action.

The following example error occurs when the `mateojackson` IAM user tries to use the console to view details about a fictional *`my-example-widget`* resource but doesn't have the fictional `cloudwatch:`*`GetWidget`* permissions.

```
User: arn:aws-cn:iam::123456789012:user/mateojackson is not authorized to perform:
 cloudwatch:GetWidget on resource: my-example-widget
```

In this case, the policy for the `mateojackson` user must be updated to allow access to the *`my-example-widget`* resource by using the `cloudwatch:`*`GetWidget`* action.

If you need help, contact your Amazon administrator. Your administrator is the person who provided you with your sign-in credentials.

I am not authorized to perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, your policies must be updated to allow you to pass a role to CloudWatch.

Some Amazon Web Services services allow you to pass an existing role to that service instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named `marymajor` tries to use the console to perform an action in CloudWatch. However, the action requires the service to have permissions that are granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws-cn:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

In this case, Mary's policies must be updated to allow her to perform the `iam:PassRole` action.

If you need help, contact your Amazon administrator. Your administrator is the person who provided you with your sign-in credentials.

I want to allow people outside of my Amazon Web Services account to access my CloudWatch resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether CloudWatch supports these features, see How Amazon CloudWatch works with IAM.
- To learn how to provide access to your resources across Amazon Web Services accounts that you own, see Providing access to an IAM user in another Amazon Web Services account that you own in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party Amazon Web Services accounts, see Providing access to Amazon Web Services accounts owned by third parties in the *IAM User Guide*.
- To learn how to provide access through identity federation, see Providing access to externally authenticated users (identity federation) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see Cross account resource access in IAM in the *IAM User Guide*.

CloudWatch dashboard permissions update

On May 1, 2018, Amazon changed the permissions required to access CloudWatch dashboards. Dashboard access in the CloudWatch console now requires permissions that were introduced in 2017 to support dashboard API operations:

- **cloudwatch:GetDashboard**
- **cloudwatch:ListDashboards**
- **cloudwatch:PutDashboard**
- **cloudwatch:DeleteDashboards**

To access CloudWatch dashboards, you need one of the following:

- The **AdministratorAccess** policy.
- The **CloudWatchFullAccess** policy.
- A custom policy that includes one or more of these specific permissions:
  - `cloudwatch:GetDashboard` and `cloudwatch:ListDashboards` to be able to view dashboards
  - `cloudwatch:PutDashboard` to be able to create or modify dashboards
  - `cloudwatch:DeleteDashboards` to be able to delete dashboards

For more information about using policies to change permissions for an IAM user, see Changing Permissions for an IAM user.

For more information about CloudWatch permissions, see Amazon CloudWatch permissions reference.

For more information about dashboard API operations, see PutDashboard in the Amazon CloudWatch API Reference.

Amazon managed (predefined) policies for CloudWatch

Amazon addresses many common use cases by providing standalone IAM policies that are created and administered by Amazon. These Amazon managed policies grant necessary permissions for common use cases so that you can avoid having to investigate what permissions are needed. For more information, see Amazon managed policies in the *IAM User Guide*.

The following Amazon managed policies, which you can attach to users in your account, are specific to CloudWatch.

Topics

CloudWatchFullAccessV2

Amazon recently added the **CloudWatchFullAccessV2** managed IAM policy. This policy grants full access to CloudWatch actions and resources and also more properly scopes the permissions granted for other services such as Amazon SNS and Amazon EC2 Auto Scaling. We recommend that you begin using this policy instead of using **CloudWatchFullAccess**. Amazon plans to deprecate **CloudWatchFullAccess** in the near future.

It includes `application-signals:` permissions so that users can access all the functionality from the CloudWatch console under Application Signals. It includes some `autoscaling:Describe` permissions so that users with this policy can see the Auto Scaling actions that are associated with CloudWatch alarms. It includes some `sns` permissions so that users with this policy can retrieve create Amazon SNS topics and associate them with CloudWatch alarms. It includes IAM permissions so that users with this policy can view information about service-linked roles associated with CloudWatch. It includes the `oam:ListSinks` and `oam:ListAttachedLinks` permissions so that users with this policy can use the console to view data shared from source accounts in CloudWatch cross-account observability.

It includes Amazon OpenSearch Service permissions to support vended logs dashboards in CloudWatch Logs, which are created with Amazon OpenSearch Service analytics.

It includes `rum`, `synthetics`, and `xray` permissions so that users can have full access to CloudWatch Synthetics, Amazon X-Ray, and CloudWatch RUM, all of which are under the CloudWatch service.

The contents of **CloudWatchFullAccessV2** are as follows:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CloudWatchFullAccessPermissions",
            "Effect": "Allow",
            "Action": [
                "application-autoscaling:DescribeScalingPolicies",
                "application-signals:*",
                "autoscaling:DescribeAutoScalingGroups",
                "autoscaling:DescribePolicies",
                "cloudwatch:*",
                "logs:*",
                "sns:CreateTopic",
                "sns:ListSubscriptions",
                "sns:ListSubscriptionsByTopic",
                "sns:ListTopics",
                "sns:Subscribe",
                "iam:GetPolicy",
                "iam:GetPolicyVersion",
                "iam:GetRole",
                "oam:ListSinks",
                "rum:*",
                "synthetics:*",
                "xray:*",
                "opensearch:ApplicationAccessAll",
                "iam:ListRoles",
                "iam:ListUsers",
                "aoss:BatchGetCollection",
                "aoss:BatchGetLifecyclePolicy",
```

```
                "es:ListApplications"
            ],
            "Resource": "*"
        },
        {
            "Sid": "CloudWatchApplicationSignalsServiceLinkedRolePermissions",
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::*:role/aws-service-role/application-
signals.cloudwatch.amazonaws.com/AWSServiceRoleForCloudWatchApplicationSignals",
            "Condition": {
                "StringLike": {
                    "iam:AWSServiceName": "application-
signals.cloudwatch.amazonaws.com"
                }
            }
        },
        {
            "Sid": "EventsServicePermissions",
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::*:role/aws-service-role/events.amazonaws.com/
AWSServiceRoleForCloudWatchEvents*",
            "Condition": {
                "StringLike": {
                    "iam:AWSServiceName": "events.amazonaws.com"
                }
            }
        },
        {
            "Sid": "OAMReadPermissions",
            "Effect": "Allow",
            "Action": [
                "oam:ListAttachedLinks"
            ],
            "Resource": "arn:aws:oam:*:*:sink/*"
        },
        {
            "Sid": "CloudWatchLogsOpenSearchCreateServiceLinkedAccess",
            "Effect": "Allow",
            "Action": [
                "iam:CreateServiceLinkedRole"
            ],
```

```
            "Resource": "arn:aws:iam::*:role/aws-service-role/
opensearchservice.amazonaws.com/AWSServiceRoleForAmazonOpenSearchService",
            "Condition": {
                "StringEquals": {
                    "iam:AWSServiceName": "opensearchservice.amazonaws.com"
                }
            }
        },
        {
            "Sid": "CloudWatchLogsObservabilityCreateServiceLinkedAccess",
            "Effect": "Allow",
            "Action": [
                "iam:CreateServiceLinkedRole"
            ],
            "Resource": "arn:aws:iam::*:role/aws-service-role/*/
AWSServiceRoleForAmazonOpenSearchServerless",
            "Condition": {
                "StringEquals": {
                    "iam:AWSServiceName": "observability.aoss.amazonaws.com"
                }
            }
        },
        {
            "Sid": "CloudWatchLogsCollectionRequestAccess",
            "Effect": "Allow",
            "Action": [
                "aoss:CreateCollection"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/CloudWatchOpenSearchIntegration": [
                        "Dashboards"
                    ]
                },
                "ForAllValues:StringEquals": {
                    "aws:TagKeys": "CloudWatchOpenSearchIntegration"
                }
            }
        },
        {
            "Sid": "CloudWatchLogsApplicationRequestAccess",
            "Effect": "Allow",
            "Action": [
```

```
                "es:CreateApplication"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/OpenSearchIntegration": [
                        "Dashboards"
                    ]
                },
                "ForAllValues:StringEquals": {
                    "aws:TagKeys": "OpenSearchIntegration"
                }
            }
        },
        {
            "Sid": "CloudWatchLogsCollectionResourceAccess",
            "Effect": "Allow",
            "Action": [
                "aoss:DeleteCollection"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/CloudWatchOpenSearchIntegration": [
                        "Dashboards"
                    ]
                }
            }
        },
        {
            "Sid": "CloudWatchLogsApplicationResourceAccess",
            "Effect": "Allow",
            "Action": [
                "es:UpdateApplication",
                "es:GetApplication"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/OpenSearchIntegration": [
                        "Dashboards"
                    ]
                }
            }
```

```
        },
        {
            "Sid": "CloudWatchLogsCollectionPolicyAccess",
            "Effect": "Allow",
            "Action": [
                "aoss:CreateSecurityPolicy",
                "aoss:CreateAccessPolicy",
                "aoss:DeleteAccessPolicy",
                "aoss:DeleteSecurityPolicy",
                "aoss:GetAccessPolicy",
                "aoss:GetSecurityPolicy",
                "aoss:APIAccessAll"
            ],
            "Resource": "*",
            "Condition": {
                "StringLike": {
                    "aoss:collection": "logs-collection-*"
                }
            }
        },
        {
            "Sid": "CloudWatchLogsIndexPolicyAccess",
            "Effect": "Allow",
            "Action": [
                "aoss:CreateAccessPolicy",
                "aoss:DeleteAccessPolicy",
                "aoss:GetAccessPolicy",
                "aoss:CreateLifecyclePolicy",
                "aoss:DeleteLifecyclePolicy"
            ],
            "Resource": "*",
            "Condition": {
                "StringLike": {
                    "aoss:index": "logs-collection-*"
                }
            }
        },
        {
            "Sid": "CloudWatchLogsStartDirectQueryAccess",
            "Effect": "Allow",
            "Action": [
                "opensearch:StartDirectQuery"
            ],
            "Resource": "arn:aws:opensearch:*:*:datasource/logs_datasource_*"
```

```
        },
        {
            "Sid": "CloudWatchLogsDQSRequestQueryAccess",
            "Effect": "Allow",
            "Action": [
                "es:AddDirectQueryDataSource"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/CloudWatchOpenSearchIntegration": [
                        "Dashboards"
                    ]
                },
                "ForAllValues:StringEquals": {
                    "aws:TagKeys": "CloudWatchOpenSearchIntegration"
                }
            }
        },
        {
            "Sid": "CloudWatchLogsDQSResourceQueryAccess",
            "Effect": "Allow",
            "Action": [
                "es:GetDirectQueryDataSource",
                "es:DeleteDirectQueryDataSource"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/CloudWatchOpenSearchIntegration": [
                        "Dashboards"
                    ]
                }
            }
        },
        {
            "Sid": "CloudWatchLogsPassRoleAccess",
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": "*",
            "Condition": {
                "StringLike": {
```

```
                    "iam:PassedToService":
"directquery.opensearchservice.amazonaws.com"
                }
            }
        },
        {
            "Sid": "CloudWatchLogsAossTagsAccess",
            "Effect": "Allow",
            "Action": [
                "aoss:TagResource",
                "es:AddTags"
            ],
            "Resource": "arn:aws:aoss:*:*:collection/*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/CloudWatchOpenSearchIntegration": [
                        "Dashboards"
                    ]
                },
                "ForAllValues:StringEquals": {
                    "aws:TagKeys": "CloudWatchOpenSearchIntegration"
                }
            }
        },
        {
            "Sid": "CloudWatchLogsEsApplicationTagsAccess",
            "Effect": "Allow",
            "Action": [
                "es:AddTags"
            ],
            "Resource": "arn:aws:opensearch:*:*:application/*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/OpenSearchIntegration": [
                        "Dashboards"
                    ]
                },
                "ForAllValues:StringEquals": {
                    "aws:TagKeys": "OpenSearchIntegration"
                }
            }
        },
        {
            "Sid": "CloudWatchLogsEsDataSourceTagsAccess",
```

```
            "Effect": "Allow",
            "Action": [
                "es:AddTags"
            ],
            "Resource": "arn:aws:opensearch:*:*:datasource/*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/CloudWatchOpenSearchIntegration": [
                        "Dashboards"
                    ]
                },
                "ForAllValues:StringEquals": {
                    "aws:TagKeys": "CloudWatchOpenSearchIntegration"
                }
            }
        }
    ]
}
```

CloudWatchFullAccess

The **CloudWatchFullAccess** policy is on the path to deprecation. We recommend that you stop using it, and use CloudWatchFullAccessV2 instead.

The contents of **CloudWatchFullAccess** are as follows:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "autoscaling:Describe*",
                "cloudwatch:*",
                "logs:*",
                "sns:*",
                "iam:GetPolicy",
                "iam:GetPolicyVersion",
                "iam:GetRole",
                "oam:ListSinks",
                "opensearch:ApplicationAccessAll",
                "iam:ListRoles",
                "iam:ListUsers",
```

```
                "aoss:BatchGetCollection",
                "aoss:BatchGetLifecyclePolicy",
                "es:ListApplications"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::*:role/aws-service-role/events.amazonaws.com/
AWSServiceRoleForCloudWatchEvents*",
            "Condition": {
                "StringLike": {
                    "iam:AWSServiceName": "events.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "oam:ListAttachedLinks"
            ],
            "Resource": "arn:aws:oam:*:*:sink/*"
        },
        {
            "Sid": "CloudWatchLogsOpenSearchCreateServiceLinkedAccess",
            "Effect": "Allow",
            "Action": [
                "iam:CreateServiceLinkedRole"
            ],
            "Resource": "arn:aws:iam::*:role/aws-service-role/
opensearchservice.amazonaws.com/AWSServiceRoleForAmazonOpenSearchService",
            "Condition": {
                "StringEquals": {
                    "iam:AWSServiceName": "opensearchservice.amazonaws.com"
                }
            }
        },
        {
            "Sid": "CloudWatchLogsObservabilityCreateServiceLinkedAccess",
            "Effect": "Allow",
            "Action": [
                "iam:CreateServiceLinkedRole"
            ],
```

```
            "Resource": "arn:aws:iam::*:role/aws-service-role/*/
AWSServiceRoleForAmazonOpenSearchServerless",
            "Condition": {
                "StringEquals": {
                    "iam:AWSServiceName": "observability.aoss.amazonaws.com"
                }
            }
        },
        {
            "Sid": "CloudWatchLogsCollectionRequestAccess",
            "Effect": "Allow",
            "Action": [
                "aoss:CreateCollection"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/CloudWatchOpenSearchIntegration": [
                        "Dashboards"
                    ]
                },
                "ForAllValues:StringEquals": {
                    "aws:TagKeys": "CloudWatchOpenSearchIntegration"
                }
            }
        },
        {
            "Sid": "CloudWatchLogsApplicationRequestAccess",
            "Effect": "Allow",
            "Action": [
                "es:CreateApplication"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/OpenSearchIntegration": [
                        "Dashboards"
                    ]
                },
                "ForAllValues:StringEquals": {
                    "aws:TagKeys": "OpenSearchIntegration"
                }
            }
        },
```

```
        {
            "Sid": "CloudWatchLogsCollectionResourceAccess",
            "Effect": "Allow",
            "Action": [
                "aoss:DeleteCollection"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/CloudWatchOpenSearchIntegration": [
                        "Dashboards"
                    ]
                }
            }
        },
        {
            "Sid": "CloudWatchLogsApplicationResourceAccess",
            "Effect": "Allow",
            "Action": [
                "es:UpdateApplication",
                "es:GetApplication"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/OpenSearchIntegration": [
                        "Dashboards"
                    ]
                }
            }
        },
        {
            "Sid": "CloudWatchLogsCollectionPolicyAccess",
            "Effect": "Allow",
            "Action": [
                "aoss:CreateSecurityPolicy",
                "aoss:CreateAccessPolicy",
                "aoss:DeleteAccessPolicy",
                "aoss:DeleteSecurityPolicy",
                "aoss:GetAccessPolicy",
                "aoss:GetSecurityPolicy",
                "aoss:APIAccessAll"
            ],
            "Resource": "*",
```

```
            "Condition": {
                "StringLike": {
                    "aoss:collection": "logs-collection-*"
                }
            }
        },
        {
            "Sid": "CloudWatchLogsIndexPolicyAccess",
            "Effect": "Allow",
            "Action": [
                "aoss:CreateAccessPolicy",
                "aoss:DeleteAccessPolicy",
                "aoss:GetAccessPolicy",
                "aoss:CreateLifecyclePolicy",
                "aoss:DeleteLifecyclePolicy"
            ],
            "Resource": "*",
            "Condition": {
                "StringLike": {
                    "aoss:index": "logs-collection-*"
                }
            }
        },
        {
            "Sid": "CloudWatchLogsStartDirectQueryAccess",
            "Effect": "Allow",
            "Action": [
                "opensearch:StartDirectQuery"
            ],
            "Resource": "arn:aws:opensearch:*:*:datasource/logs_datasource_*"
        },
        {
            "Sid": "CloudWatchLogsDQSRequestQueryAccess",
            "Effect": "Allow",
            "Action": [
                "es:AddDirectQueryDataSource"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/CloudWatchOpenSearchIntegration": [
                        "Dashboards"
                    ]
                },
```

```
                "ForAllValues:StringEquals": {
                    "aws:TagKeys": "CloudWatchOpenSearchIntegration"
                }
            }
        },
        {
            "Sid": "CloudWatchLogsDQSResourceQueryAccess",
            "Effect": "Allow",
            "Action": [
                "es:GetDirectQueryDataSource",
                "es:DeleteDirectQueryDataSource"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/CloudWatchOpenSearchIntegration": [
                        "Dashboards"
                    ]
                }
            }
        },
        {
            "Sid": "CloudWatchLogsPassRoleAccess",
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": "*",
            "Condition": {
                "StringLike": {
                    "iam:PassedToService":
 "directquery.opensearchservice.amazonaws.com"
                }
            }
        },
        {
            "Sid": "CloudWatchLogsAossTagsAccess",
            "Effect": "Allow",
            "Action": [
                "aoss:TagResource",
                "es:AddTags"
            ],
            "Resource": "arn:aws:aoss:*:*:collection/*",
            "Condition": {
```

```
                "StringEquals": {
                    "aws:ResourceTag/CloudWatchOpenSearchIntegration": [
                        "Dashboards"
                    ]
                },
                "ForAllValues:StringEquals": {
                    "aws:TagKeys": "CloudWatchOpenSearchIntegration"
                }
            }
        },
        {
            "Sid": "CloudWatchLogsEsApplicationTagsAccess",
            "Effect": "Allow",
            "Action": [
                "es:AddTags"
            ],
            "Resource": "arn:aws:opensearch:*:*:application/*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/OpenSearchIntegration": [
                        "Dashboards"
                    ]
                },
                "ForAllValues:StringEquals": {
                    "aws:TagKeys": "OpenSearchIntegration"
                }
            }
        },
        {
            "Sid": "CloudWatchLogsEsDataSourceTagsAccess",
            "Effect": "Allow",
            "Action": [
                "es:AddTags"
            ],
            "Resource": "arn:aws:opensearch:*:*:datasource/*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/CloudWatchOpenSearchIntegration": [
                        "Dashboards"
                    ]
                },
                "ForAllValues:StringEquals": {
                    "aws:TagKeys": "CloudWatchOpenSearchIntegration"
                }
```

```
            }
        }
    ]
}
```

CloudWatchReadOnlyAccess

The **CloudWatchReadOnlyAccess** policy grants read-only access to CloudWatch.

The policy includes some `logs:` permissions, so that users with this policy can use the console to view CloudWatch Logs information and CloudWatch Logs Insights queries. It includes `autoscaling:Describe*`, so that users with this policy can see the Auto Scaling actions that are associated with CloudWatch alarms. It includes the `application-signals:` permissions so that users can use Application Signals to monitor the health of their services. It includes `application-autoscaling:DescribeScalingPolicies`, so that users with this policy can access information about Application Auto Scaling policies. It includes `sns:Get*` and `sns:List*`, so that users with this policy can retrieve information about the Amazon SNS topics that receive notifications about CloudWatch alarms. It includes the `oam:ListSinks` and `oam:ListAttachedLinks` permissions, so that users with this policy can use the console to view data shared from source accounts in CloudWatch cross-account observability. It includes the `iam:GetRole` permissions so that users can check if CloudWatch Application Signals have been set up.

It includes `rum`, `synthetics`, and `xray` permissions so that users can have read-only access to CloudWatch Synthetics, Amazon X-Ray, and CloudWatch RUM, all of which are under the CloudWatch service.

The following is the content of the **CloudWatchReadOnlyAccess** policy.

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CloudWatchReadOnlyAccessPermissions",
            "Effect": "Allow",
            "Action": [
                "application-autoscaling:DescribeScalingPolicies",
                "application-signals:BatchGet*",
                "application-signals:Get*",
                "application-signals:List*",
                "autoscaling:Describe*",
                "cloudwatch:BatchGet*",
```

```
                "cloudwatch:Describe*",
                "cloudwatch:GenerateQuery",
                "cloudwatch:Get*",
                "cloudwatch:List*",
                "logs:Get*",
                "logs:List*",
                "logs:StartQuery",
                "logs:StopQuery",
                "logs:Describe*",
                "logs:TestMetricFilter",
                "logs:FilterLogEvents",
                "logs:StartLiveTail",
                "logs:StopLiveTail",
                "oam:ListSinks",
                "sns:Get*",
                "sns:List*",
                "rum:BatchGet*",
                "rum:Get*",
                "rum:List*",
                "synthetics:Describe*",
                "synthetics:Get*",
                "synthetics:List*",
                "xray:BatchGet*",
                "xray:Get*",
                "xray:List*",
                "xray:StartTraceRetrieval",
                "xray:CancelTraceRetrieval"
            ],
            "Resource": "*"
        },
        {
            "Sid": "OAMReadPermissions",
            "Effect": "Allow",
            "Action": [
                "oam:ListAttachedLinks"
            ],
            "Resource": "arn:aws:oam:*:*:sink/*"
        },
        {
            "Sid": "CloudWatchReadOnlyGetRolePermissions",
            "Effect": "Allow",
            "Action": "iam:GetRole",
            "Resource": "arn:aws:iam::*:role/aws-service-role/application-
signals.cloudwatch.amazonaws.com/AWSServiceRoleForCloudWatchApplicationSignals"
```

```
        }
      ]
}
```

CloudWatchActionsEC2Access

The **CloudWatchActionsEC2Access** policy grants read-only access to CloudWatch alarms and metrics in addition to Amazon EC2 metadata. It also grants access to the Stop, Terminate, and Reboot API actions for EC2 instances.

The following is the content of the **CloudWatchActionsEC2Access** policy.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "cloudwatch:Describe*",
        "ec2:Describe*",
        "ec2:RebootInstances",
        "ec2:StopInstances",
        "ec2:TerminateInstances"
      ],
      "Resource": "*"
    }
  ]
}
```

CloudWatch-CrossAccountAccess

The **CloudWatch-CrossAccountAccess** managed policy is used by the **CloudWatch-CrossAccountSharingRole** IAM role. This role and policy enable users of cross-account dashboards to view automatic dashboards in each account that is sharing dashboards.

The following is the content of **CloudWatch-CrossAccountAccess**:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
```

```
            "Action": [
                "sts:AssumeRole"
            ],
            "Resource": [
                "arn:aws:iam::*:role/CloudWatch-CrossAccountSharing*"
            ],
            "Effect": "Allow"
        }
    ]
}
```

CloudWatchAutomaticDashboardsAccess

The **CloudWatchAutomaticDashboardsAccess** managed policy grants access to CloudWatch for non-CloudWatch APIs, so that resources such as Lambda functions can be displayed on CloudWatch automatic dashboards.

The following is the content of **CloudWatchAutomaticDashboardsAccess**:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "autoscaling:DescribeAutoScalingGroups",
        "cloudfront:GetDistribution",
        "cloudfront:ListDistributions",
        "dynamodb:DescribeTable",
        "dynamodb:ListTables",
        "ec2:DescribeInstances",
        "ec2:DescribeVolumes",
        "ecs:DescribeClusters",
        "ecs:DescribeContainerInstances",
        "ecs:ListClusters",
        "ecs:ListContainerInstances",
        "ecs:ListServices",
        "elasticache:DescribeCacheClusters",
        "elasticbeanstalk:DescribeEnvironments",
        "elasticfilesystem:DescribeFileSystems",
        "elasticloadbalancing:DescribeLoadBalancers",
        "kinesis:DescribeStream",
        "kinesis:ListStreams",
        "lambda:GetFunction",
```

```
        "lambda:ListFunctions",
        "rds:DescribeDBClusters",
        "rds:DescribeDBInstances",
        "resource-groups:ListGroupResources",
        "resource-groups:ListGroups",
        "route53:GetHealthCheck",
        "route53:ListHealthChecks",
        "s3:ListAllMyBuckets",
        "s3:ListBucket",
        "sns:ListTopics",
        "sqs:GetQueueAttributes",
        "sqs:GetQueueUrl",
        "sqs:ListQueues",
        "synthetics:DescribeCanariesLastRun",
        "tag:GetResources"
      ],
      "Effect": "Allow",
      "Resource": "*"
    },
    {
      "Action": [
        "apigateway:GET"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:apigateway:*::/restapis*"
      ]
    }
  ]
```

CloudWatchAgentServerPolicy

The **CloudWatchAgentServerPolicy** policy can be used in IAM roles that are attached to Amazon EC2 instances to allow the CloudWatch agent to read information from the instance and write it to CloudWatch. Its contents are as follows.

CloudWatchAgentAdminPolicy

The **CloudWatchAgentAdminPolicy** policy can be used in IAM roles that are attached to Amazon EC2 instances. This policy allows the CloudWatch agent to read information from the instance and write it to CloudWatch, and also to write information to Parameter Store. Its contents are as follows.

> **Note**
>
> You can review these permissions policies by signing in to the IAM console and searching for specific policies there.

You can also create your own custom IAM policies to allow permissions for CloudWatch actions and resources. You can attach these custom policies to the IAM users or groups that require those permissions.

Amazon managed (predefined) policies for CloudWatch cross-account observability

The policies in this section grant permissions related to CloudWatch cross-account observability. For more information, see CloudWatch cross-account observability.

CloudWatchCrossAccountSharingConfiguration

The **CloudWatchCrossAccountSharingConfiguration** policy grants access to create, manage, and view Observability Access Manager links for sharing CloudWatch resources between accounts. For more information, see CloudWatch cross-account observability. The contents are as follows:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:Link",
                "oam:ListLinks"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "oam:DeleteLink",
                "oam:GetLink",
                "oam:TagResource"
            ],
            "Resource": "arn:aws:oam:*:*:link/*"
```

```
        },
        {
            "Effect": "Allow",
            "Action": [
                "oam:CreateLink",
                "oam:UpdateLink"
            ],
            "Resource": [
                "arn:aws:oam:*:*:link/*",
                "arn:aws:oam:*:*:sink/*"
            ]
        }
    ]
}
```

OAMFullAccess

The **OAMFullAccess** policy grants access to create, manage, and view Observability Access Manager sinks and links, which are used for CloudWatch cross-account observability.

The **OAMFullAccess** policy by itself does not permit you to share observability data across links. To create a link to share CloudWatch metrics, you also need either **CloudWatchFullAccess** or **CloudWatchCrossAccountSharingConfiguration**. To create a link to share CloudWatch Logs log groups, you also need either **CloudWatchLogsFullAccess** or **CloudWatchLogsCrossAccountSharingConfiguration**. To create a link to share X-Ray traces, you also need either **AWSXRayFullAccess** or **AWSXRayCrossAccountSharingConfiguration**.

For more information, see CloudWatch cross-account observability. The contents are as follows:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "oam:*"
            ],
            "Resource": "*"
        }
    ]
}
```

OAMReadOnlyAccess

The **OAMReadOnlyAccess** policy grants read-only access to Observability Access Manager resources, which are used for CloudWatch cross-account observability. For more information, see CloudWatch cross-account observability. The contents are as follows:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "oam:Get*",
                "oam:List*"
            ],
            "Resource": "*"
        }
    ]
}
```

Amazon managed (predefined) policies for Amazon Q Developer operational investigations

The policies in this section grant permissions related to Amazon Q Developer operational investigations. For more information, see Amazon Q Developer operational investigations (Preview).

AIOpsConsoleAdminPolicy

The **AIOpsConsoleAdminPolicy** policy grants full access to all Amazon Q Developer operational investigations actions and their required permissions via the Amazon console. This policy also grants limited access to other service's APIs required for Amazon Q Developer operational investigations functionality.

- The `aiops` permissions grant access to all Amazon Q Developer operational investigations actions.
- The `organizations`, `sso`, `identitystore`, and `sts` permissions allow actions needed for IAM Identity Center management which facilitate identity-aware sessions.
- The `ssm` permissions are required for SSM Ops Item integration with third-party issue management.

- The `iam` permissions are needed so that administrators can pass IAM roles to the `aiops` and `ssm.integrations` services, and the role is later used by the assistant to analyze Amazon resources

> **Important**
>
> These permissions allow users with this policy to pass any IAM role to the `aiops` and `ssm.integrations` services.

- It allows APIs from services outside Amazon Q Developer operational investigations, which are required for investigation feature functionality. These include actions to configure Amazon Q Developer in chat applications, Amazon KMS, CloudTrail trails, and SSM third-party issue management.

The following are the contents of the **AIOpsConsoleAdminPolicy** policy.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AIOpsAdmin",
      "Effect": "Allow",
      "Action": [
        "aiops:*"
      ],
      "Resource": "*"
    },
    {
      "Sid": "OrganizationsAccess",
      "Effect": "Allow",
      "Action": [
        "organizations:ListAWSServiceAccessForOrganization",
        "organizations:DescribeOrganization"
      ],
      "Resource": "*"
    },
    {
      "Sid": "SSOApplicationManagement",
      "Effect": "Allow",
      "Action": [
        "sso:PutApplicationAccessScope",
```

```
        "sso:PutApplicationAssignmentConfiguration",
        "sso:PutApplicationGrant",
        "sso:PutApplicationAuthenticationMethod",
        "sso:DeleteApplication"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:CalledViaLast": "aiops.amazonaws.com",
          "aws:ResourceTag/ManagedByAmazonAIOperations": "true"
        }
      }
    },
    {
      "Sid": "SSOApplicationTagManagement",
      "Effect": "Allow",
      "Action": [
        "sso:CreateApplication",
        "sso:TagResource"
      ],
      "Resource": [
        "arn:aws:sso:::instance/*",
        "arn:aws:sso::aws:applicationProvider/aiops"
      ],
      "Condition": {
        "StringEquals": {
          "aws:CalledViaLast": "aiops.amazonaws.com",
          "aws:RequestTag/ManagedByAmazonAIOperations": "true"
        },
        "ForAllValues:StringEquals": {
          "aws:TagKeys": [
            "ManagedByAmazonAIOperations"
          ]
        }
      }
    },
    {
      "Sid": "SSOTagManagement",
      "Effect": "Allow",
      "Action": [
        "sso:TagResource"
      ],
      "Resource": "arn:aws:sso::*:application/*",
      "Condition": {
```

```
          "StringEquals": {
            "aws:CalledViaLast": "aiops.amazonaws.com",
            "aws:ResourceTag/ManagedByAmazonAIOperations": "true"
          },
          "ForAllValues:StringEquals": {
            "aws:TagKeys": [
              "ManagedByAmazonAIOperations"
            ]
          }
        }
      },
      {
        "Sid": "SSOManagementAccess",
        "Effect": "Allow",
        "Action": [
          "identitystore:DescribeUser",
          "sso:ListApplications",
          "sso:ListInstances",
          "sso:DescribeRegisteredRegions",
          "sso:GetSharedSsoConfiguration",
          "sso:DescribeInstance",
          "sso:GetSSOStatus",
          "sso-directory:DescribeUsers"
        ],
        "Resource": "*"
      },
      {
        "Sid": "AllowSTSContextSetting",
        "Effect": "Allow",
        "Action": [
          "sts:SetContext"
        ],
        "Resource": "arn:aws:sts::*:self"
      },
      {
        "Sid": "IdentityPropagationAccess",
        "Effect": "Allow",
        "Action": [
          "signin:ListTrustedIdentityPropagationApplicationsForConsole"
        ],
        "Resource": "*"
      },
      {
        "Sid": "CloudtrailAccess",
```

```
      "Effect": "Allow",
      "Action": [
        "cloudtrail:ListTrails",
        "cloudtrail:DescribeTrails",
        "cloudtrail:ListEventDataStores"
      ],
      "Resource": "*"
    },
    {
      "Sid": "KMSAccess",
      "Effect": "Allow",
      "Action": [
        "kms:ListAliases"
      ],
      "Resource": "*"
    },
    {
      "Sid": "SSMIntegrationSecretsManagerAccess",
      "Effect": "Allow",
      "Action": [
        "secretsmanager:CreateSecret",
        "secretsmanager:PutResourcePolicy",
        "secretsmanager:UpdateSecret",
        "secretsmanager:DeleteSecret"
      ],
      "Resource": "arn:aws:secretsmanager:*:*:secret:aws/ssm/3p/*"
    },
    {
      "Sid": "SSMIntegrationAccess",
      "Effect": "Allow",
      "Action": [
        "ssm:GetServiceSetting",
        "ssm:UpdateServiceSetting"
      ],
      "Resource": "arn:aws:ssm:*:*:servicesetting/integrations/*"
    },
    {
      "Sid": "SSMIntegrationCreatePolicy",
      "Effect": "Allow",
      "Action": [
        "iam:CreatePolicy"
      ],
      "Resource": "arn:aws:iam::*:policy/service-role/
AWSServiceRoleSSMIntegrationsPolicy*"
```

```
    },
    {
      "Sid": "ChatbotConfigurations",
      "Effect": "Allow",
      "Action": [
        "chatbot:DescribeChimeWebhookConfigurations",
        "chatbot:DescribeSlackWorkspaces",
        "chatbot:DescribeSlackChannelConfigurations",
        "chatbot:ListMicrosoftTeamsChannelConfigurations",
        "chatbot:ListMicrosoftTeamsConfiguredTeams"
      ],
      "Resource": "*"
    },
    {
      "Sid": "IAMPassRoleToAIOps",
      "Effect": "Allow",
      "Action": [
        "iam:PassRole"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "iam:PassedToService": "aiops.amazonaws.com"
        }
      }
    },
    {
      "Sid": "IAMListRoles",
      "Effect": "Allow",
      "Action": [
        "iam:ListRoles"
      ],
      "Resource": "*"
    },
    {
      "Sid": "TagBoundaryPermission",
      "Effect": "Allow",
      "Action": [
        "tag:GetTagKeys"
      ],
      "Resource": "*"
    },
    {
      "Sid": "IAMPassRoleToSSMIntegration",
```

```
      "Effect": "Allow",
      "Action": [
        "iam:PassRole"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "iam:PassedToService": "ssm.integrations.amazonaws.com"
        },
        "ArnEquals": {
          "iam:AssociatedResourceArn": "arn:aws:aiops:*:*:investigation-group/*"
        }
      }
    },
    {
      "Sid": "SSMOpsItemAccess",
      "Effect": "Allow",
      "Action": [
        "ssm:CreateOpsItem",
        "ssm:AddTagsToResource"
      ],
      "Resource": "arn:*:ssm:*:*:opsitem/*",
      "Condition": {
        "StringEquals": {
          "aws:RequestTag/Integration": "CloudWatch",
          "aws:ResourceTag/Integration": "CloudWatch"
        },
        "ForAllValues:StringEquals": {
          "aws:TagKeys": [
            "Integration"
          ]
        }
      }
    }
  ]
}
```

AIOpsOperatorAccess

The **AIOpsOperatorAccess** policy grants access to a limited set of Amazon Q Developer operational investigations APIs which include creating, updating, and deleting investigations, investigation events, and investigation resources.

This policy only provides access to investigations. You should be sure that IAM principals with this policy also have permissions to read CloudWatch observability data such as metrics, SLOs, and CloudWatch Logs query results.

- The `aiops` permissions allow access to Amazon Q Developer operational investigations APIs to create, update, and delete investigations.
- The `sso`, `identitystore`, and `sts` permssions allow actions needed for IAM Identity Center management which facilitate identity-aware sessions.
- The `ssm` permissions are required for SSM Ops Item integration with third-party issue management.

The following are the contents of the **AIOpsOperatorAccess** policy.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AIOpsOperatorAccess",
            "Effect": "Allow",
            "Action": [
                "aiops:CreateInvestigation",
                "aiops:CreateInvestigationEvent",
                "aiops:CreateInvestigationResource",
                "aiops:DeleteInvestigation",
                "aiops:Get*",
                "aiops:List*",
                "aiops:UpdateInvestigation",
                "aiops:UpdateInvestigationEvent"
            ],
            "Resource": "*"
        },
        {
            "Sid": "SSOManagementAccess",
            "Effect": "Allow",
            "Action": [
                "identitystore:DescribeUser",
                "sso:DescribeInstance",
                "sso-directory:DescribeUsers"
            ],
            "Resource": "*"
        },
```

```
        {
            "Sid": "AllowSTSContextSetting",
            "Effect": "Allow",
            "Action": [
                "sts:SetContext"
            ],
            "Resource": "arn:aws:sts::*:self"
        },
        {
            "Sid": "SSMSettingServiceIntegration",
            "Effect": "Allow",
            "Action": [
                "ssm:GetServiceSetting"
            ],
            "Resource": "arn:aws:ssm:*:*:servicesetting/integrations/*"
        },
        {
            "Sid": "SSMIntegrationTagAccess",
            "Effect": "Allow",
            "Action": [
                "ssm:AddTagsToResource",
                "ssm:CreateOpsItem"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/Integration": [
                        "CloudWatch"
                    ]
                },
                "ForAllValues:StringEquals": {
                    "aws:TagKeys": "Integration"
                }
            }
        },
        {
            "Sid": "SSMOpsItemIntegration",
            "Effect": "Allow",
            "Action": [
                "ssm:DeleteOpsItem",
                "ssm:UpdateOpsItem"
            ],
            "Resource": "*",
            "Condition": {
```

```
                "StringEquals": {
                    "aws:ResourceTag/Integration": [
                        "CloudWatch"
                    ]
                }
            }
        },
        {
            "Sid": "SSMTagOperation",
            "Effect": "Allow",
            "Action": [
                "ssm:AddTagsToResource"
            ],
            "Resource": "arn:aws:ssm:*:*:opsitem/*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/Integration": [
                        "CloudWatch"
                    ]
                },
                "ForAllValues:StringEquals": {
                    "aws:TagKeys": "Integration"
                }
            }
        },
        {
            "Sid": "SSMOpsSummaryIntegration",
            "Effect": "Allow",
            "Action": [
                "ssm:GetOpsSummary"
            ],
            "Resource": "*"
        }
    ]
}
```

AIOpsReadOnlyAccess

The **AIOpsReadOnlyAccess** policy grants read-only permissions for Amazon Q investigations and other related services.

The following are the contents of the **AIOpsReadOnlyAccess** policy.

```
{
```

```
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AIOpsReadOnlyAccess",
      "Effect": "Allow",
      "Action": [
        "aiops:Get*",
        "aiops:List*"
      ],
      "Resource": "*"
    },
    {
      "Sid": "SSOManagementAccess",
      "Effect": "Allow",
      "Action": [
        "identitystore:DescribeUser",
        "sso:DescribeInstance",
        "sso-directory:DescribeUsers"
      ],
      "Resource": "*"
    }
  ]
}
```

IAM policy for Amazon Q Developer operational investigations (AIOpsAssistantPolicy)

The **AIOpsAssistantPolicy** policy in this section is not to be assigned to users or administrators. Instead, you assign **AIOpsAssistantPolicy** to Amazon Q Developer to enable it to analyze your Amazon resources during investigations of operational events. This policy is scoped based on the resources that Amazon Q Developer supports during investigations, and will be updated as more resources are supported.

You can also choose to assign the general Amazon ReadOnlyAccess to the assistant in addition to assigning it **AIOpsAssistantPolicy**. The reason to do this is that **ReadOnlyAccess** will be updated more frequently by Amazon with permissions for new Amazon services and actions that are released. The **AIOpsAssistantPolicy** will also be updated for new actions, but not as frequently.

The following are the contents of the **AIOpsAssistantPolicy** policy.

```
{
  "Version": "2012-10-17",
  "Statement": [
```

```
    {
      "Sid": "AIOPSServiceAccess",
      "Effect": "Allow",
      "Action": [
        "access-analyzer:GetAnalyzer",
        "access-analyzer:List*",
        "acm-pca:Describe*",
        "acm-pca:GetCertificate",
        "acm-pca:GetCertificateAuthorityCertificate",
        "acm-pca:GetCertificateAuthorityCsr",
        "acm-pca:List*",
        "acm:DescribeCertificate",
        "acm:GetAccountConfiguration",
        "airflow:List*",
        "amplify:GetApp",
        "amplify:GetBranch",
        "amplify:GetDomainAssociation",
        "amplify:List*",
        "aoss:BatchGetCollection",
        "aoss:BatchGetLifecyclePolicy",
        "aoss:BatchGetVpcEndpoint",
        "aoss:GetAccessPolicy",
        "aoss:GetSecurityConfig",
        "aoss:GetSecurityPolicy",
        "aoss:List*",
        "appconfig:GetApplication",
        "appconfig:GetConfigurationProfile",
        "appconfig:GetEnvironment",
        "appconfig:GetHostedConfigurationVersion",
        "appconfig:List*",
        "appflow:Describe*",
        "appflow:List*",
        "application-autoscaling:Describe*",
        "application-signals:BatchGetServiceLevelObjectiveBudgetReport",
        "application-signals:GetService",
        "application-signals:GetServiceLevelObjective",
        "application-signals:List*",
        "applicationinsights:Describe*",
        "applicationinsights:List*",
        "apprunner:Describe*",
        "apprunner:List*",
        "appstream:Describe*",
        "appstream:List*",
        "appsync:GetApiAssociation",
```

```
            "appsync:GetDomainName",
            "appsync:GetFunction",
            "appsync:GetResolver",
            "appsync:GetSourceApiAssociation",
            "appsync:List*",
            "aps:Describe*",
            "aps:List*",
            "arc-zonal-shift:GetManagedResource",
            "arc-zonal-shift:List*",
            "athena:GetCapacityAssignmentConfiguration",
            "athena:GetCapacityReservation",
            "athena:GetDataCatalog",
            "athena:GetNamedQuery",
            "athena:GetPreparedStatement",
            "athena:GetWorkGroup",
            "athena:List*",
            "auditmanager:GetAssessment",
            "auditmanager:List*",
            "autoscaling:Describe*",
            "backup-gateway:GetHypervisor",
            "backup-gateway:List*",
            "backup:Describe*",
            "backup:GetBackupPlan",
            "backup:GetBackupSelection",
            "backup:GetBackupVaultAccessPolicy",
            "backup:GetBackupVaultNotifications",
            "backup:GetRestoreTestingPlan",
            "backup:GetRestoreTestingSelection",
            "backup:List*",
            "batch:DescribeComputeEnvironments",
            "batch:DescribeJobQueues",
            "batch:DescribeSchedulingPolicies",
            "batch:List*",
            "bedrock:GetAgent",
            "bedrock:GetAgentActionGroup",
            "bedrock:GetAgentAlias",
            "bedrock:GetAgentKnowledgeBase",
            "bedrock:GetDataSource",
            "bedrock:GetGuardrail",
            "bedrock:GetKnowledgeBase",
            "bedrock:List*",
            "budgets:Describe*",
            "budgets:List*",
            "ce:Describe*",
```

```
            "ce:GetAnomalyMonitors",
            "ce:GetAnomalySubscriptions",
            "ce:List*",
            "chatbot:Describe*",
            "chatbot:GetMicrosoftTeamsChannelConfiguration",
            "chatbot:List*",
            "cleanrooms-ml:GetTrainingDataset",
            "cleanrooms-ml:List*",
            "cleanrooms:GetAnalysisTemplate",
            "cleanrooms:GetCollaboration",
            "cleanrooms:GetConfiguredTable",
            "cleanrooms:GetConfiguredTableAnalysisRule",
            "cleanrooms:GetConfiguredTableAssociation",
            "cleanrooms:GetMembership",
            "cleanrooms:List*",
            "cloudformation:Describe*",
            "cloudformation:GetResource",
            "cloudformation:GetStackPolicy",
            "cloudformation:GetTemplate",
            "cloudformation:List*",
            "cloudfront:Describe*",
            "cloudfront:GetCachePolicy",
            "cloudfront:GetCloudFrontOriginAccessIdentity",
            "cloudfront:GetContinuousDeploymentPolicy",
            "cloudfront:GetDistribution",
            "cloudfront:GetDistributionConfig",
            "cloudfront:GetFunction",
            "cloudfront:GetKeyGroup",
            "cloudfront:GetMonitoringSubscription",
            "cloudfront:GetOriginAccessControl",
            "cloudfront:GetOriginRequestPolicy",
            "cloudfront:GetPublicKey",
            "cloudfront:GetRealtimeLogConfig",
            "cloudfront:GetResponseHeadersPolicy",
            "cloudfront:List*",
            "cloudtrail:Describe*",
            "cloudtrail:GetChannel",
            "cloudtrail:GetEventDataStore",
            "cloudtrail:GetEventSelectors",
            "cloudtrail:GetInsightSelectors",
            "cloudtrail:GetQueryResults",
            "cloudtrail:GetResourcePolicy",
            "cloudtrail:GetTrail",
            "cloudtrail:GetTrailStatus",
```

```
            "cloudtrail:List*",
            "cloudtrail:LookupEvents",
            "cloudtrail:StartQuery",
            "cloudwatch:Describe*",
            "cloudwatch:GenerateQuery",
            "cloudwatch:GetDashboard",
            "cloudwatch:GetInsightRuleReport",
            "cloudwatch:GetMetricData",
            "cloudwatch:GetMetricStream",
            "cloudwatch:GetService",
            "cloudwatch:GetServiceLevelObjective",
            "cloudwatch:List*",
            "codeartifact:Describe*",
            "codeartifact:GetDomainPermissionsPolicy",
            "codeartifact:GetRepositoryPermissionsPolicy",
            "codeartifact:List*",
            "codebuild:BatchGetFleets",
            "codebuild:List*",
            "codecommit:GetRepository",
            "codecommit:GetRepositoryTriggers",
            "codedeploy:BatchGetDeployments",
            "codedeploy:BatchGetDeploymentTargets",
            "codedeploy:GetApplication",
            "codedeploy:GetDeploymentConfig",
            "codedeploy:List*",
            "codeguru-profiler:Describe*",
            "codeguru-profiler:GetNotificationConfiguration",
            "codeguru-profiler:GetPolicy",
            "codeguru-profiler:List*",
            "codeguru-reviewer:Describe*",
            "codeguru-reviewer:List*",
            "codepipeline:GetPipeline",
            "codepipeline:GetPipelineState",
            "codepipeline:List*",
            "codestar-connections:GetConnection",
            "codestar-connections:GetRepositoryLink",
            "codestar-connections:GetSyncConfiguration",
            "codestar-connections:List*",
            "codestar-notifications:Describe*",
            "codestar-notifications:List*",
            "cognito-identity:DescribeIdentityPool",
            "cognito-identity:GetIdentityPoolRoles",
            "cognito-identity:ListIdentityPools",
            "cognito-identity:ListTagsForResource",
```

```
            "cognito-idp:AdminListGroupsForUser",
            "cognito-idp:DescribeIdentityProvider",
            "cognito-idp:DescribeResourceServer",
            "cognito-idp:DescribeRiskConfiguration",
            "cognito-idp:DescribeUserImportJob",
            "cognito-idp:DescribeUserPool",
            "cognito-idp:DescribeUserPoolDomain",
            "cognito-idp:GetGroup",
            "cognito-idp:GetLogDeliveryConfiguration",
            "cognito-idp:GetUICustomization",
            "cognito-idp:GetUserPoolMfaConfig",
            "cognito-idp:GetWebACLForResource",
            "cognito-idp:ListGroups",
            "cognito-idp:ListIdentityProviders",
            "cognito-idp:ListResourceServers",
            "cognito-idp:ListUserPoolClients",
            "cognito-idp:ListUserPools",
            "cognito-idp:ListUsers",
            "cognito-idp:ListTagsForResource",
            "comprehend:Describe*",
            "comprehend:List*",
            "config:Describe*",
            "config:GetStoredQuery",
            "config:List*",
            "connect:Describe*",
            "connect:GetTaskTemplate",
            "connect:List*",
            "databrew:Describe*",
            "databrew:List*",
            "datapipeline:Describe*",
            "datapipeline:GetPipelineDefinition",
            "datapipeline:List*",
            "datasync:Describe*",
            "datasync:List*",
            "deadline:GetFarm",
            "deadline:GetFleet",
            "deadline:GetLicenseEndpoint",
            "deadline:GetMonitor",
            "deadline:GetQueue",
            "deadline:GetQueueEnvironment",
            "deadline:GetQueueFleetAssociation",
            "deadline:GetStorageProfile",
            "deadline:List*",
            "detective:GetMembers",
```

```
            "detective:List*",
            "devicefarm:GetDevicePool",
            "devicefarm:GetInstanceProfile",
            "devicefarm:GetNetworkProfile",
            "devicefarm:GetProject",
            "devicefarm:GetTestGridProject",
            "devicefarm:GetVPCEConfiguration",
            "devicefarm:List*",
            "devops-guru:Describe*",
            "devops-guru:GetResourceCollection",
            "devops-guru:List*",
            "dms:Describe*",
            "dms:List*",
            "ds:Describe*",
            "dynamodb:Describe*",
            "dynamodb:GetResourcePolicy",
            "dynamodb:List*",
            "ec2:Describe*",
            "ec2:GetAssociatedEnclaveCertificateIamRoles",
            "ec2:GetIpamPoolAllocations",
            "ec2:GetIpamPoolCidrs",
            "ec2:GetManagedPrefixListEntries",
            "ec2:GetNetworkInsightsAccessScopeContent",
            "ec2:GetSnapshotBlockPublicAccessState",
            "ec2:GetTransitGatewayMulticastDomainAssociations",
            "ec2:GetTransitGatewayRouteTableAssociations",
            "ec2:GetTransitGatewayRouteTablePropagations",
            "ec2:GetVerifiedAccessEndpointPolicy",
            "ec2:GetVerifiedAccessGroupPolicy",
            "ec2:GetVerifiedAccessInstanceWebAcl",
            "ec2:SearchLocalGatewayRoutes",
            "ec2:SearchTransitGatewayRoutes",
            "ecr:Describe*",
            "ecr:GetLifecyclePolicy",
            "ecr:GetRegistryPolicy",
            "ecr:GetRepositoryPolicy",
            "ecr:List*",
            "ecs:Describe*",
            "ecs:List*",
            "eks:Describe*",
            "eks:List*",
            "elastic-inference:Describe*",
            "elasticache:Describe*",
            "elasticache:List*",
```

```
            "elasticbeanstalk:Describe*",
            "elasticbeanstalk:List*",
            "elasticfilesystem:Describe*",
            "elasticloadbalancing:Describe*",
            "elasticmapreduce:Describe*",
            "elasticmapreduce:List*",
            "emr-containers:Describe*",
            "emr-containers:List*",
            "emr-serverless:GetApplication",
            "emr-serverless:List*",
            "es:Describe*",
            "es:List*",
            "events:Describe*",
            "events:List*",
            "evidently:GetExperiment",
            "evidently:GetFeature",
            "evidently:GetLaunch",
            "evidently:GetProject",
            "evidently:GetSegment",
            "evidently:List*",
            "firehose:Describe*",
            "firehose:List*",
            "fis:GetExperimentTemplate",
            "fis:GetTargetAccountConfiguration",
            "fis:List*",
            "fms:GetNotificationChannel",
            "fms:GetPolicy",
            "fms:List*",
            "forecast:Describe*",
            "forecast:List*",
            "frauddetector:BatchGetVariable",
            "frauddetector:Describe*",
            "frauddetector:GetDetectors",
            "frauddetector:GetDetectorVersion",
            "frauddetector:GetEntityTypes",
            "frauddetector:GetEventTypes",
            "frauddetector:GetExternalModels",
            "frauddetector:GetLabels",
            "frauddetector:GetListElements",
            "frauddetector:GetListsMetadata",
            "frauddetector:GetModelVersion",
            "frauddetector:GetOutcomes",
            "frauddetector:GetRules",
            "frauddetector:GetVariables",
```

```
            "frauddetector:List*",
            "fsx:Describe*",
            "gamelift:Describe*",
            "gamelift:List*",
            "globalaccelerator:Describe*",
            "globalaccelerator:List*",
            "glue:GetDatabase",
            "glue:GetDatabases",
            "glue:GetJob",
            "glue:GetRegistry",
            "glue:GetSchema",
            "glue:GetSchemaVersion",
            "glue:GetTable",
            "glue:GetTags",
            "glue:GetTrigger",
            "glue:List*",
            "glue:querySchemaVersionMetadata",
            "grafana:Describe*",
            "grafana:List*",
            "greengrass:Describe*",
            "greengrass:GetDeployment",
            "greengrass:List*",
            "groundstation:GetConfig",
            "groundstation:GetDataflowEndpointGroup",
            "groundstation:GetMissionProfile",
            "groundstation:List*",
            "guardduty:GetDetector",
            "guardduty:GetFilter",
            "guardduty:GetIPSet",
            "guardduty:GetMalwareProtectionPlan",
            "guardduty:GetMasterAccount",
            "guardduty:GetMembers",
            "guardduty:GetThreatIntelSet",
            "guardduty:List*",
            "health:DescribeEvents",
            "health:DescribeEventDetails",
            "healthlake:Describe*",
            "healthlake:List*",
            "iam:GetGroup",
            "iam:GetGroupPolicy",
            "iam:GetInstanceProfile",
            "iam:GetLoginProfile",
            "iam:GetOpenIDConnectProvider",
            "iam:GetPolicy",
```

```
            "iam:GetPolicyVersion",
            "iam:GetRole",
            "iam:GetRolePolicy",
            "iam:GetSAMLProvider",
            "iam:GetServerCertificate",
            "iam:GetServiceLinkedRoleDeletionStatus",
            "iam:GetUser",
            "iam:GetUserPolicy",
            "iam:ListOpenIDConnectProviders",
            "iam:ListServerCertificates",
            "iam:ListVirtualMFADevices",
            "identitystore:DescribeGroup",
            "identitystore:DescribeGroupMembership",
            "identitystore:ListGroupMemberships",
            "identitystore:ListGroups",
            "imagebuilder:GetComponent",
            "imagebuilder:GetContainerRecipe",
            "imagebuilder:GetDistributionConfiguration",
            "imagebuilder:GetImage",
            "imagebuilder:GetImagePipeline",
            "imagebuilder:GetImageRecipe",
            "imagebuilder:GetInfrastructureConfiguration",
            "imagebuilder:GetLifecyclePolicy",
            "imagebuilder:GetWorkflow",
            "imagebuilder:List*",
            "inspector2:List*",
            "inspector:Describe*",
            "inspector:List*",
            "internetmonitor:GetMonitor",
            "internetmonitor:List*",
            "iot:Describe*",
            "iot:GetPackage",
            "iot:GetPackageVersion",
            "iot:GetPolicy",
            "iot:GetThingShadow",
            "iot:GetTopicRule",
            "iot:GetTopicRuleDestination",
            "iot:GetV2LoggingOptions",
            "iot:List*",
            "iotanalytics:Describe*",
            "iotanalytics:List*",
            "iotevents:Describe*",
            "iotevents:List*",
            "iotfleethub:Describe*",
```

```
            "iotfleethub:List*",
            "iotsitewise:Describe*",
            "iotsitewise:List*",
            "iotwireless:GetDestination",
            "iotwireless:GetDeviceProfile",
            "iotwireless:GetFuotaTask",
            "iotwireless:GetMulticastGroup",
            "iotwireless:GetNetworkAnalyzerConfiguration",
            "iotwireless:GetServiceProfile",
            "iotwireless:GetWirelessDevice",
            "iotwireless:GetWirelessGateway",
            "iotwireless:GetWirelessGatewayTaskDefinition",
            "iotwireless:List*",
            "ivs:GetChannel",
            "ivs:GetEncoderConfiguration",
            "ivs:GetPlaybackRestrictionPolicy",
            "ivs:GetRecordingConfiguration",
            "ivs:GetStage",
            "ivs:List*",
            "ivschat:GetLoggingConfiguration",
            "ivschat:GetRoom",
            "ivschat:List*",
            "kafka:Describe*",
            "kafka:GetClusterPolicy",
            "kafka:List*",
            "kafkaconnect:Describe*",
            "kafkaconnect:List*",
            "kendra:Describe*",
            "kendra:List*",
            "kinesis:Describe*",
            "kinesis:List*",
            "kinesisanalytics:Describe*",
            "kinesisanalytics:List*",
            "kinesisvideo:Describe*",
            "kms:DescribeKey",
            "kms:ListResourceTags",
            "kms:ListKeys",
            "lakeformation:Describe*",
            "lakeformation:GetLFTag",
            "lakeformation:GetResourceLFTags",
            "lakeformation:List*",
            "lambda:GetAlias",
            "lambda:GetCodeSigningConfig",
            "lambda:GetEventSourceMapping",
```

```
            "lambda:GetFunction",
            "lambda:GetFunctionCodeSigningConfig",
            "lambda:GetFunctionConfiguration",
            "lambda:GetFunctionEventInvokeConfig",
            "lambda:GetFunctionRecursionConfig",
            "lambda:GetFunctionUrlConfig",
            "lambda:GetLayerVersion",
            "lambda:GetLayerVersionPolicy",
            "lambda:GetPolicy",
            "lambda:GetProvisionedConcurrencyConfig",
            "lambda:GetRuntimeManagementConfig",
            "lambda:List*",
            "launchwizard:GetDeployment",
            "launchwizard:List*",
            "lex:Describe*",
            "lex:List*",
            "license-manager:GetLicense",
            "license-manager:List*",
            "lightsail:GetAlarms",
            "lightsail:GetBuckets",
            "lightsail:GetCertificates",
            "lightsail:GetContainerServices",
            "lightsail:GetDisk",
            "lightsail:GetDisks",
            "lightsail:GetInstance",
            "lightsail:GetInstances",
            "lightsail:GetLoadBalancer",
            "lightsail:GetLoadBalancers",
            "lightsail:GetLoadBalancerTlsCertificates",
            "lightsail:GetStaticIp",
            "lightsail:GetStaticIps",
            "logs:Describe*",
            "logs:FilterLogEvents",
            "logs:GetDataProtectionPolicy",
            "logs:GetDelivery",
            "logs:GetDeliveryDestination",
            "logs:GetDeliveryDestinationPolicy",
            "logs:GetDeliverySource",
            "logs:GetLogAnomalyDetector",
            "logs:GetLogDelivery",
            "logs:GetQueryResults",
            "logs:List*",
            "logs:StartQuery",
            "logs:StopLiveTail",
```

```
        "logs:StopQuery",
        "logs:TestMetricFilter",
        "lookoutmetrics:Describe*",
        "lookoutmetrics:List*",
        "lookoutvision:Describe*",
        "lookoutvision:List*",
        "m2:GetApplication",
        "m2:GetEnvironment",
        "m2:List*",
        "macie2:GetAllowList",
        "macie2:GetCustomDataIdentifier",
        "macie2:GetFindingsFilter",
        "macie2:GetMacieSession",
        "macie2:List*",
        "mediaconnect:Describe*",
        "mediaconnect:List*",
        "medialive:Describe*",
        "medialive:GetCloudWatchAlarmTemplate",
        "medialive:GetCloudWatchAlarmTemplateGroup",
        "medialive:GetEventBridgeRuleTemplate",
        "medialive:GetEventBridgeRuleTemplateGroup",
        "medialive:GetSignalMap",
        "medialive:List*",
        "mediapackage-vod:Describe*",
        "mediapackage-vod:List*",
        "mediapackage:Describe*",
        "mediapackage:List*",
        "mediapackagev2:GetChannel",
        "mediapackagev2:GetChannelGroup",
        "mediapackagev2:GetChannelPolicy",
        "mediapackagev2:GetOriginEndpoint",
        "mediapackagev2:GetOriginEndpointPolicy",
        "mediapackagev2:List*",
        "memorydb:Describe*",
        "memorydb:List*",
        "mobiletargeting:GetInAppTemplate",
        "mobiletargeting:List*",
        "mq:Describe*",
        "mq:List*",
        "network-firewall:Describe*",
        "network-firewall:List*",
        "networkmanager:Describe*",
        "networkmanager:GetConnectAttachment",
        "networkmanager:GetConnectPeer",
```

```
            "networkmanager:GetCoreNetwork",
            "networkmanager:GetCoreNetworkPolicy",
            "networkmanager:GetCustomerGatewayAssociations",
            "networkmanager:GetDevices",
            "networkmanager:GetLinkAssociations",
            "networkmanager:GetLinks",
            "networkmanager:GetSites",
            "networkmanager:GetSiteToSiteVpnAttachment",
            "networkmanager:GetTransitGatewayPeering",
            "networkmanager:GetTransitGatewayRegistrations",
            "networkmanager:GetTransitGatewayRouteTableAttachment",
            "networkmanager:GetVpcAttachment",
            "networkmanager:List*",
            "nimble:GetLaunchProfile",
            "nimble:GetStreamingImage",
            "nimble:GetStudio",
            "nimble:GetStudioComponent",
            "nimble:List*",
            "oam:GetLink",
            "oam:GetSink",
            "oam:GetSinkPolicy",
            "oam:List*",
            "omics:GetAnnotationStore",
            "omics:GetReferenceStore",
            "omics:GetRunGroup",
            "omics:GetSequenceStore",
            "omics:GetVariantStore",
            "omics:GetWorkflow",
            "omics:List*",
            "opsworks-cm:Describe*",
            "opsworks-cm:List*",
            "organizations:Describe*",
            "organizations:List*",
            "osis:GetPipeline",
            "osis:List*",
            "payment-cryptography:GetAlias",
            "payment-cryptography:GetKey",
            "payment-cryptography:List*",
            "pca-connector-ad:GetConnector",
            "pca-connector-ad:GetDirectoryRegistration",
            "pca-connector-ad:GetServicePrincipalName",
            "pca-connector-ad:GetTemplate",
            "pca-connector-ad:GetTemplateGroupAccessControlEntry",
            "pca-connector-ad:List*",
```

```
        "pca-connector-scep:GetChallengeMetadata",
        "pca-connector-scep:GetConnector",
        "pca-connector-scep:List*",
        "personalize:Describe*",
        "personalize:List*",
        "pipes:Describe*",
        "pipes:List*",
        "proton:GetEnvironmentTemplate",
        "proton:GetServiceTemplate",
        "proton:List*",
        "qbusiness:GetApplication",
        "qbusiness:GetDataSource",
        "qbusiness:GetIndex",
        "qbusiness:GetPlugin",
        "qbusiness:GetRetriever",
        "qbusiness:GetWebExperience",
        "qbusiness:List*",
        "qldb:Describe*",
        "qldb:List*",
        "ram:GetPermission",
        "ram:List*",
        "rds:Describe*",
        "rds:List*",
        "redshift-serverless:GetNamespace",
        "redshift-serverless:GetWorkgroup",
        "redshift-serverless:List*",
        "redshift:Describe*",
        "refactor-spaces:GetApplication",
        "refactor-spaces:GetEnvironment",
        "refactor-spaces:GetRoute",
        "refactor-spaces:List*",
        "rekognition:Describe*",
        "rekognition:List*",
        "resiliencehub:Describe*",
        "resiliencehub:List*",
        "resource-explorer-2:GetDefaultView",
        "resource-explorer-2:GetIndex",
        "resource-explorer-2:GetView",
        "resource-explorer-2:List*",
        "resource-groups:GetGroup",
        "resource-groups:GetGroupConfiguration",
        "resource-groups:GetGroupQuery",
        "resource-groups:GetTags",
        "resource-groups:List*",
```

```
            "robomaker:Describe*",
            "robomaker:List*",
            "route53-recovery-control-config:Describe*",
            "route53-recovery-control-config:List*",
            "route53-recovery-readiness:GetCell",
            "route53-recovery-readiness:GetReadinessCheck",
            "route53-recovery-readiness:GetRecoveryGroup",
            "route53-recovery-readiness:GetResourceSet",
            "route53-recovery-readiness:List*",
            "route53:GetDNSSEC",
            "route53:GetHealthCheck",
            "route53:GetHostedZone",
            "route53:List*",
            "route53profiles:GetProfile",
            "route53profiles:GetProfileAssociation",
            "route53profiles:GetProfileResourceAssociation",
            "route53profiles:List*",
            "route53resolver:GetFirewallDomainList",
            "route53resolver:GetFirewallRuleGroup",
            "route53resolver:GetFirewallRuleGroupAssociation",
            "route53resolver:GetOutpostResolver",
            "route53resolver:GetResolverConfig",
            "route53resolver:GetResolverQueryLogConfig",
            "route53resolver:GetResolverQueryLogConfigAssociation",
            "route53resolver:GetResolverRule",
            "route53resolver:GetResolverRuleAssociation",
            "route53resolver:List*",
            "rum:GetAppMonitor",
            "rum:List*",
            "s3-outposts:GetAccessPoint",
            "s3-outposts:GetAccessPointPolicy",
            "s3-outposts:GetBucket",
            "s3-outposts:GetBucketPolicy",
            "s3-outposts:GetBucketTagging",
            "s3-outposts:GetLifecycleConfiguration",
            "s3-outposts:List*",
            "s3:GetAccelerateConfiguration",
            "s3:GetAccessGrant",
            "s3:GetAccessGrantsInstance",
            "s3:GetAccessGrantsLocation",
            "s3:GetAccessPoint",
            "s3:GetAccessPointConfigurationForObjectLambda",
            "s3:GetAccessPointForObjectLambda",
            "s3:GetAccessPointPolicy",
```

```
            "s3:GetAccessPointPolicyForObjectLambda",
            "s3:GetAccessPointPolicyStatusForObjectLambda",
            "s3:GetAnalyticsConfiguration",
            "s3:GetBucketAcl",
            "s3:GetBucketCORS",
            "s3:GetBucketLocation",
            "s3:GetBucketLogging",
            "s3:GetBucketNotification",
            "s3:GetBucketObjectLockConfiguration",
            "s3:GetBucketOwnershipControls",
            "s3:GetBucketPolicy",
            "s3:GetBucketPublicAccessBlock",
            "s3:GetBucketTagging",
            "s3:GetBucketVersioning",
            "S3:GetBucketWebsite",
            "s3:GetEncryptionConfiguration",
            "s3:GetIntelligentTieringConfiguration",
            "s3:GetInventoryConfiguration",
            "s3:GetLifecycleConfiguration",
            "s3:GetMetricsConfiguration",
            "s3:GetMultiRegionAccessPoint",
            "s3:GetMultiRegionAccessPointPolicy",
            "s3:GetMultiRegionAccessPointPolicyStatus",
            "s3:GetReplicationConfiguration",
            "s3:GetStorageLensConfiguration",
            "s3:GetStorageLensConfigurationTagging",
            "s3:GetStorageLensGroup",
            "s3:List*",
            "sagemaker:Describe*",
            "sagemaker:List*",
            "scheduler:GetSchedule",
            "scheduler:GetScheduleGroup",
            "scheduler:List*",
            "schemas:Describe*",
            "schemas:GetResourcePolicy",
            "schemas:List*",
            "secretsmanager:Describe*",
            "secretsmanager:GetResourcePolicy",
            "secretsmanager:List*",
            "securityhub:BatchGetAutomationRules",
            "securityhub:BatchGetSecurityControls",
            "securityhub:Describe*",
            "securityhub:GetConfigurationPolicy",
            "securityhub:GetConfigurationPolicyAssociation",
```

```
            "securityhub:GetEnabledStandards",
            "securityhub:GetFindingAggregator",
            "securityhub:GetInsights",
            "securityhub:List*",
            "securitylake:GetSubscriber",
            "securitylake:List*",
            "servicecatalog:Describe*",
            "servicecatalog:GetApplication",
            "servicecatalog:GetAttributeGroup",
            "servicecatalog:List*",
            "servicequotas:GetServiceQuota",
            "ses:Describe*",
            "ses:GetAccount",
            "ses:GetAddonInstance",
            "ses:GetAddonSubscription",
            "ses:GetArchive",
            "ses:GetConfigurationSet",
            "ses:GetConfigurationSetEventDestinations",
            "ses:GetContactList",
            "ses:GetDedicatedIpPool",
            "ses:GetDedicatedIps",
            "ses:GetEmailIdentity",
            "ses:GetEmailTemplate",
            "ses:GetIngressPoint",
            "ses:GetRelay",
            "ses:GetRuleSet",
            "ses:GetTemplate",
            "ses:GetTrafficPolicy",
            "ses:List*",
            "shield:Describe*",
            "shield:List*",
            "signer:GetSigningProfile",
            "signer:List*",
            "sns:GetDataProtectionPolicy",
            "sns:GetSubscriptionAttributes",
            "sns:GetTopicAttributes",
            "sns:List*",
            "sqs:GetQueueAttributes",
            "sqs:GetQueueUrl",
            "sqs:List*",
            "ssm-contacts:GetContact",
            "ssm-contacts:GetContactChannel",
            "ssm-contacts:List*",
            "ssm-incidents:GetReplicationSet",
```

```
            "ssm-incidents:GetResponsePlan",
            "ssm-incidents:List*",
            "ssm-sap:GetApplication",
            "ssm-sap:List*",
            "ssm:Describe*",
            "ssm:GetDefaultPatchBaseline",
            "ssm:GetDocument",
            "ssm:GetParameters",
            "ssm:GetPatchBaseline",
            "ssm:GetResourcePolicies",
            "ssm:List*",
            "sso-directory:SearchGroups",
            "sso-directory:SearchUsers",
            "sso:GetInlinePolicyForPermissionSet",
            "sso:GetManagedApplicationInstance",
            "sso:GetPermissionsBoundaryForPermissionSet",
            "sso:GetSharedSsoConfiguration",
            "sso:ListAccountAssignments",
            "sso:ListApplicationAssignments",
            "sso:ListApplications",
            "sso:ListCustomerManagedPolicyReferencesInPermissionSet",
            "sso:ListInstances",
            "sso:ListManagedPoliciesInPermissionSet",
            "sso:ListTagsForResource",
            "states:Describe*",
            "states:List*",
            "synthetics:Describe*",
            "synthetics:GetCanary",
            "synthetics:GetGroup",
            "synthetics:List*",
            "tag:GetResources",
            "timestream:Describe*",
            "timestream:List*",
            "transfer:Describe*",
            "transfer:List*",
            "verifiedpermissions:GetIdentitySource",
            "verifiedpermissions:GetPolicy",
            "verifiedpermissions:GetPolicyStore",
            "verifiedpermissions:GetPolicyTemplate",
            "verifiedpermissions:GetSchema",
            "verifiedpermissions:List*",
            "vpc-lattice:GetAccessLogSubscription",
            "vpc-lattice:GetAuthPolicy",
            "vpc-lattice:GetListener",
```

```
        "vpc-lattice:GetResourcePolicy",
        "vpc-lattice:GetRule",
        "vpc-lattice:GetService",
        "vpc-lattice:GetServiceNetwork",
        "vpc-lattice:GetServiceNetworkServiceAssociation",
        "vpc-lattice:GetServiceNetworkVpcAssociation",
        "vpc-lattice:GetTargetGroup",
        "vpc-lattice:List*",
        "wafv2:GetIPSet",
        "wafv2:GetLoggingConfiguration",
        "wafv2:GetRegexPatternSet",
        "wafv2:GetRuleGroup",
        "wafv2:GetWebACL",
        "wafv2:GetWebACLForResource",
        "wafv2:List*",
        "workspaces-web:GetBrowserSettings",
        "workspaces-web:GetIdentityProvider",
        "workspaces-web:GetNetworkSettings",
        "workspaces-web:GetPortal",
        "workspaces-web:GetPortalServiceProviderMetadata",
        "workspaces-web:GetTrustStore",
        "workspaces-web:GetUserAccessLoggingSettings",
        "workspaces-web:GetUserSettings",
        "workspaces-web:List*",
        "workspaces:Describe*",
        "xray:BatchGetTraces",
        "xray:GetGroup",
        "xray:GetGroups",
        "xray:GetSamplingRules",
        "xray:GetServiceGraph",
        "xray:GetTraceSummaries",
        "xray:List*"
      ],
      "Resource": "*"
    },
    {
      "Sid": "AIOPSS3AccessForAmplify",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:GetObjectVersion",
        "s3:GetObjectAcl"
      ],
      "Resource": [
```

```
        "arn:aws:s3:::amplify",
        "arn:aws:s3:::cdk--assets--*"
      ],
      "Condition": {
        "StringEquals": {
          "aws:ViaAWSService": [
            "amplify.amazonaws.com"
          ],
          "aws:PrincipalAccount": [
            "${aws:ResourceAccount}"
          ]
        }
      }
    },
    {
      "Sid": "AIOPSAPIGatewayAccess",
      "Effect": "Allow",
      "Action": [
        "apigateway:GET"
      ],
      "Resource": [
        "arn:aws:apigateway:*::/restapis",
        "arn:aws:apigateway:*::/restapis/*",
        "arn:aws:apigateway:*::/restapis/*/deployments",
        "arn:aws:apigateway:*::/restapis/*/deployments/*",
        "arn:aws:apigateway:*::/restapis/*/resources/*/methods/*/integrations",
        "arn:aws:apigateway:*::/restapis/*/resources/*/methods/*/integrations/*",
        "arn:aws:apigateway:*::/restapis/*/stages",
        "arn:aws:apigateway:*::/restapis/*/stages/*",
        "arn:aws:apigateway:*::/apis",
        "arn:aws:apigateway:*::/apis/*",
        "arn:aws:apigateway:*::/apis/*/deployments",
        "arn:aws:apigateway:*::/apis/*/deployments/*",
        "arn:aws:apigateway:*::/apis/*/integrations",
        "arn:aws:apigateway:*::/apis/*/integrations/*",
        "arn:aws:apigateway:*::/apis/*/stages",
        "arn:aws:apigateway:*::/apis/*/stages/*"
      ]
    }
  ]
}
```

Amazon managed (predefined) policies for CloudWatch Application Signals

The policies in this section grant permissions related to CloudWatch Application Signals. For more information, see Application Signals.

CloudWatchApplicationSignalsReadOnlyAccess

Amazon has added the **CloudWatchApplicationSignalsReadOnlyAccess** managed IAM policy. This policy grants read only access to actions and resources available to users in the CloudWatch console under Application Signals. It includes `application-signals:` policies so that users can use CloudWatch Application signals to view, investigate and monitor the health of their services. It includes an `iam:GetRole` policy to allow users to retrieve information about an IAM role. It includes `logs:` policies to start and stop queries, retrieve the configuration for a metruc filter, and obtain query results. It includes `cloudwatch:` policies so that users can obtain information about a CloudWatch alarm or metrics. It includes `synthetics:` policies so that users can retrieve information about synthetics canary runs. It includes `rum:` policies to run batch operations, retrieve data, and update metrics definitions for RUM clients. It includes an `xray:` policy to retrieve trace summaries.

The following are the contents of the **CloudWatchApplicationSignalsReadOnlyAccess** policy.

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Sid": "CloudWatchApplicationSignalsReadOnlyAccessPermissions",
   "Effect": "Allow",
   "Action": [
    "application-signals:BatchGetServiceLevelObjectiveBudgetReport",
    "application-signals:GetService",
    "application-signals:GetServiceLevelObjective",
    "application-signals:ListServiceLevelObjectives",
    "application-signals:ListServiceDependencies",
    "application-signals:ListServiceDependents",
    "application-signals:ListServiceOperations",
    "application-signals:ListServices",
    "application-signals:ListTagsForResource"
   ],
   "Resource": "*"
  },
  {
   "Sid": "CloudWatchApplicationSignalsGetRolePermissions",
```

```
    "Effect": "Allow",
    "Action": "iam:GetRole",
    "Resource": "arn:aws:iam::*:role/aws-service-role/application-
signals.cloudwatch.amazonaws.com/AWSServiceRoleForCloudWatchApplicationSignals"
   },
   {
    "Sid": "CloudWatchApplicationSignalsLogGroupPermissions",
    "Effect": "Allow",
    "Action": [
     "logs:StartQuery"
    ],
    "Resource": "arn:aws:logs:*:*:log-group:/aws/application-signals/data:*"
   },
   {
    "Sid": "CloudWatchApplicationSignalsLogsPermissions",
    "Effect": "Allow",
    "Action": [
     "logs:StopQuery",
     "logs:GetQueryResults"
    ],
    "Resource": "*"
   },
   {
    "Sid": "CloudWatchApplicationSignalsAlarmsReadPermissions",
    "Effect": "Allow",
    "Action": [
     "cloudwatch:DescribeAlarms"
    ],
    "Resource": "*"
   },
   {
    "Sid": "CloudWatchApplicationSignalsMetricsReadPermissions",
    "Effect": "Allow",
    "Action": [
     "cloudwatch:GetMetricData",
     "cloudwatch:ListMetrics"
    ],
    "Resource": "*"
   },
   {
    "Sid": "CloudWatchApplicationSignalsSyntheticsReadPermissions",
    "Effect": "Allow",
    "Action": [
     "synthetics:DescribeCanaries",
```

```
      "synthetics:DescribeCanariesLastRun",
      "synthetics:GetCanaryRuns"
    ],
    "Resource": "*"
   },
   {
    "Sid": "CloudWatchApplicationSignalsRumReadPermissions",
    "Effect": "Allow",
    "Action": [
      "rum:BatchGetRumMetricDefinitions",
      "rum:GetAppMonitor",
      "rum:GetAppMonitorData",
      "rum:ListAppMonitors"
    ],
    "Resource": "*"
   },
   {
    "Sid": "CloudWatchApplicationSignalsXrayReadPermissions",
    "Effect": "Allow",
    "Action": [
      "xray:GetTraceSummaries"
    ],
    "Resource": "*"
   }
  ]
}
```

CloudWatchApplicationSignalsFullAccess

Amazon has added the **CloudWatchApplicationSignalsFullAccess** managed IAM policy. This policy grants access to all actions and resources available to users in the CloudWatch console. It includes `application-signals:` policies so that users can use CloudWatch Application signals to view, investigate and monitor the health of their services. It uses `cloudwatch:` policies to retrieve data from metrics and alarms. It uses `logs:` policies to manage queries and filters. It uses `synthetics:` policies so that users can retrieve information about synthetics canary runs. It includes `rum:` policies to run batch operations, retrieve data and update metrics definitions for RUM clients. It includes an `xray:` policy to retrieve trace summaries. It includes `arn:aws:cloudwatch:*:*:alarm:` policies so that users can retrieve information about a service level objective (SLO) alarm. It includes `iam:` policies to manage IAM roles. It uses `sns:` policies to create, list, and subscribe to an Amazon SNS topic.

The following are the contents of the **CloudWatchApplicationSignalsFullAccess** policy.

```
{
 "Version": "2012-10-17",
 "Statement": [
    {
        "Sid": "CloudWatchApplicationSignalsFullAccessPermissions",
        "Effect": "Allow",
        "Action": "application-signals:*",
        "Resource": "*"
    },
    {
        "Sid": "CloudWatchApplicationSignalsAlarmsPermissions",
        "Effect": "Allow",
        "Action": "cloudwatch:DescribeAlarms",
        "Resource": "*"
    },
    {
        "Sid": "CloudWatchApplicationSignalsMetricsPermissions",
        "Effect": "Allow",
        "Action": [
            "cloudwatch:GetMetricData",
            "cloudwatch:ListMetrics"
        ],
        "Resource": "*"
    },
    {
        "Sid": "CloudWatchApplicationSignalsLogGroupPermissions",
        "Effect": "Allow",
        "Action": [
            "logs:StartQuery"
        ],
        "Resource": "arn:aws:logs:*:*:log-group:/aws/application-signals/data:*"
    },
    {
        "Sid": "CloudWatchApplicationSignalsLogsPermissions",
        "Effect": "Allow",
        "Action": [
            "logs:StopQuery",
            "logs:GetQueryResults"
        ],
        "Resource": "*"
    },
    {
        "Sid": "CloudWatchApplicationSignalsSyntheticsPermissions",
```

```
            "Effect": "Allow",
            "Action": [
                "synthetics:DescribeCanaries",
                "synthetics:DescribeCanariesLastRun",
                "synthetics:GetCanaryRuns"
            ],
            "Resource": "*"
        },
        {
            "Sid": "CloudWatchApplicationSignalsRumPermissions",
            "Effect": "Allow",
            "Action": [
                "rum:BatchCreateRumMetricDefinitions",
                "rum:BatchDeleteRumMetricDefinitions",
                "rum:BatchGetRumMetricDefinitions",
                "rum:GetAppMonitor",
                "rum:GetAppMonitorData",
                "rum:ListAppMonitors",
                "rum:PutRumMetricsDestination",
                "rum:UpdateRumMetricDefinition"
            ],
            "Resource": "*"
        },
        {
            "Sid": "CloudWatchApplicationSignalsXrayPermissions",
            "Effect": "Allow",
            "Action": "xray:GetTraceSummaries",
            "Resource": "*"
        },
        {
            "Sid": "CloudWatchApplicationSignalsPutMetricAlarmPermissions",
            "Effect": "Allow",
            "Action": "cloudwatch:PutMetricAlarm",
            "Resource": [
                "arn:aws:cloudwatch:*:*:alarm:SLO-AttainmentGoalAlarm-*",
                "arn:aws:cloudwatch:*:*:alarm:SLO-WarningAlarm-*",
                "arn:aws:cloudwatch:*:*:alarm:SLI-HealthAlarm-*"
            ]
        },
        {
            "Sid": "CloudWatchApplicationSignalsCreateServiceLinkedRolePermissions",
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
```

```
        "Resource": "arn:aws:iam::*:role/aws-service-role/application-
signals.cloudwatch.amazonaws.com/AWSServiceRoleForCloudWatchApplicationSignals",
        "Condition": {
            "StringLike": {
                "iam:AWSServiceName": "application-signals.cloudwatch.amazonaws.com"
            }
        }
    },
    {
        "Sid": "CloudWatchApplicationSignalsGetRolePermissions",
        "Effect": "Allow",
        "Action": "iam:GetRole",
        "Resource": "arn:aws:iam::*:role/aws-service-role/application-
signals.cloudwatch.amazonaws.com/AWSServiceRoleForCloudWatchApplicationSignals"
    },
    {
        "Sid": "CloudWatchApplicationSignalsSnsWritePermissions",
        "Effect": "Allow",
        "Action": [
            "sns:CreateTopic",
            "sns:Subscribe"
        ],
        "Resource": "arn:aws:sns:*:*:cloudwatch-application-signals-*"
    },
    {
        "Sid": "CloudWatchApplicationSignalsSnsReadPermissions",
        "Effect": "Allow",
        "Action": "sns:ListTopics",
        "Resource": "*"
    }
 ]
}
```

CloudWatchLambdaApplicationSignalsExecutionRolePolicy

This policy is used when CloudWatch Application Signals is enabled for Lambda workloads. It enables write access to X-Ray and the log group used by CloudWatch Application Signals.

The following are the contents of the **CloudWatchLambdaApplicationSignalsExecutionRolePolicy** policy.

```
{
 "Version": "2012-10-17",
```

```
  "Statement": [
   {
    "Sid": "CloudWatchApplicationSignalsXrayWritePermissions",
    "Effect": "Allow",
    "Action": [
     "xray:PutTraceSegments"
    ],
    "Resource": [
     "*"
    ],
    "Condition": {
     "StringEquals": {
      "aws:ResourceAccount": "${aws:PrincipalAccount}"
     }
    }
   },
   {
    "Sid": "CloudWatchApplicationSignalsLogGroupWritePermissions",
    "Effect": "Allow",
    "Action": [
     "logs:CreateLogGroup",
     "logs:CreateLogStream",
     "logs:PutLogEvents"
    ],
    "Resource": "arn:aws:logs:*:*:log-group:/aws/application-signals/data:*",
    "Condition": {
     "StringEquals": {
      "aws:ResourceAccount": "${aws:PrincipalAccount}"
     }
    }
   }
  ]
}
```

Amazon managed (predefined) policies for CloudWatch Synthetics

The **CloudWatchSyntheticsFullAccess** and **CloudWatchSyntheticsReadOnlyAccess** Amazon managed policies are available for you to assign to users who will manage or use CloudWatch Synthetics. The following additional policies are also relevant:

- **AmazonS3ReadOnlyAccess** and **CloudWatchReadOnlyAccess –** These are necessary to read all Synthetics data in the CloudWatch console.

- **AWSLambdaReadOnlyAccess –** Required to view the source code used by canaries.
- **CloudWatchSyntheticsFullAccess –** Allows you to create canaries. Additionally, to create and delete canaries that have a new IAM role created for them, you need the following inline policy statement:

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "synthetics:*"
         ],
         "Resource":"*"
      },
      {
         "Effect":"Allow",
         "Action":[
            "s3:CreateBucket",
            "s3:PutEncryptionConfiguration"
         ],
         "Resource":[
            "arn:aws:s3:::cw-syn-results-*"
         ]
      },
      {
         "Effect":"Allow",
         "Action":[
            "iam:ListRoles",
            "s3:ListAllMyBuckets",
            "s3:GetBucketLocation",
            "xray:GetTraceSummaries",
            "xray:BatchGetTraces",
            "apigateway:GET"
         ],
         "Resource":"*"
      },
      {
         "Effect":"Allow",
         "Action":[
            "s3:GetObject",
            "s3:ListBucket"
         ],
```

```
            "Resource":"arn:aws:s3:::cw-syn-*"
         },
         {
            "Effect":"Allow",
            "Action":[
               "s3:GetObjectVersion"
            ],
            "Resource":"arn:aws:s3:::aws-synthetics-library-*"
         },
         {
            "Effect":"Allow",
            "Action":[
               "iam:PassRole"
            ],
            "Resource":[
               "arn:aws:iam::*:role/service-role/CloudWatchSyntheticsRole*"
            ],
            "Condition":{
               "StringEquals":{
                  "iam:PassedToService":[
                     "lambda.amazonaws.com",
                     "synthetics.amazonaws.com"
                  ]
               }
            }
         },
         {
            "Effect":"Allow",
            "Action":[
               "iam:GetRole"
            ],
            "Resource":[
               "arn:aws:iam::*:role/service-role/CloudWatchSyntheticsRole*"
            ]
         },
         {
            "Effect":"Allow",
            "Action":[
               "cloudwatch:GetMetricData",
               "cloudwatch:GetMetricStatistics"
            ],
            "Resource":"*"
         },
         {
```

```
         "Effect":"Allow",
         "Action":[
            "cloudwatch:PutMetricAlarm",
            "cloudwatch:DeleteAlarms"
         ],
         "Resource":[
            "arn:aws:cloudwatch:*:*:alarm:Synthetics-*"
         ]
      },
      {
         "Effect":"Allow",
         "Action":[
            "cloudwatch:DescribeAlarms"
         ],
         "Resource":[
            "arn:aws:cloudwatch:*:*:alarm:*"
         ]
      },
      {
         "Effect":"Allow",
         "Action":[
            "lambda:CreateFunction",
            "lambda:AddPermission",
            "lambda:PublishVersion",
            "lambda:UpdateFunctionConfiguration",
            "lambda:GetFunctionConfiguration"
         ],
         "Resource":[
            "arn:aws:lambda:*:*:function:cwsyn-*"
         ]
      },
      {
          "Effect": "Allow",
          "Action": [
            "lambda:GetLayerVersion",
            "lambda:PublishLayerVersion",
            "lambda:DeleteLayerVersion"
          ],
         "Resource": [
            "arn:aws:lambda:*:*:layer:cwsyn-*",
            "arn:aws:lambda:*:*:layer:Synthetics:*",
            "arn:aws:lambda:*:*:layer:Synthetics_Selenium:*",
            "arn:aws:lambda:*:*:layer:AWS-CW-Synthetics*:*"
         ]
```

```
      },
      {
         "Effect":"Allow",
         "Action":[
            "ec2:DescribeVpcs",
            "ec2:DescribeSubnets",
            "ec2:DescribeSecurityGroups"
         ],
         "Resource":[
            "*"
         ]
      },
      {
         "Effect":"Allow",
         "Action":[
            "sns:ListTopics"
         ],
         "Resource":[
            "*"
         ]
      },
      {
         "Effect":"Allow",
         "Action":[
            "sns:CreateTopic",
            "sns:Subscribe",
            "sns:ListSubscriptionsByTopic"
         ],
         "Resource":[
            "arn:*:sns:*:*:Synthetics-*"
         ]
      }
   ]
}
```

Important

Granting a user the `iam:CreateRole`, `iam:DeleteRole`, `iam:CreatePolicy`, `iam:DeletePolicy`, `iam:AttachRolePolicy`, and `iam:DetachRolePolicy` permissions gives that user full administrative access to create, attach, and delete roles and policies that have ARNs that match `arn:aws:iam::*:role/service-role/CloudWatchSyntheticsRole*` and `arn:aws:iam::*:policy/service-role/`

CloudWatchSyntheticsPolicy*. For example, a user with these permissions can create a policy that has full permissions for all resources, and attach that policy to any role that matches that ARN pattern. Be careful to whom you grant these permissions.

For information about attaching policies and granting permissions to users, see Changing Permissions for an IAM User and To embed an inline policy for a user or role.

CloudWatchSyntheticsFullAccess

The following is the content of the **CloudWatchSyntheticsFullAccess** policy.

```
{
    "Version": "2012-10-17",
    "Statement": [{
            "Effect": "Allow",
            "Action": [
                "synthetics:*"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:CreateBucket",
                "s3:PutEncryptionConfiguration"
            ],
            "Resource": [
                "arn:aws:s3:::cw-syn-results-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:ListRoles",
                "s3:ListAllMyBuckets",
                "xray:GetTraceSummaries",
                "xray:BatchGetTraces",
                "apigateway:GET"
            ],
            "Resource": "*"
        },
```

```
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketLocation"
            ],
            "Resource": "arn:aws:s3:::*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:ListBucket"
            ],
            "Resource": "arn:aws:s3:::cw-syn-*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObjectVersion"
            ],
            "Resource": "arn:aws:s3:::aws-synthetics-library-*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": [
                "arn:aws:iam::*:role/service-role/CloudWatchSyntheticsRole*"
            ],
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": [
                        "lambda.amazonaws.com",
                        "synthetics.amazonaws.com"
                    ]
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:GetRole",
                "iam:ListAttachedRolePolicies"
```

```
            ],
            "Resource": [
                "arn:aws:iam::*:role/service-role/CloudWatchSyntheticsRole*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:GetMetricData",
                "cloudwatch:GetMetricStatistics"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:PutMetricAlarm",
                "cloudwatch:DeleteAlarms"
            ],
            "Resource": [
                "arn:aws:cloudwatch:*:*:alarm:Synthetics-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:DescribeAlarms"
            ],
            "Resource": [
                "arn:aws:cloudwatch:*:*:alarm:*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:GetLogRecord",
                "logs:DescribeLogStreams",
                "logs:StartQuery",
                "logs:GetLogEvents",
                "logs:FilterLogEvents",
                "logs:GetLogGroupFields"
            ],
            "Resource": [
                "arn:aws:logs:*:*:log-group:/aws/lambda/cwsyn-*"
```

```
            ],
            "Condition": {
                "StringEquals": {
                    "aws:ResourceAccount": "${aws:PrincipalAccount}"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "lambda:CreateFunction",
                "lambda:AddPermission",
                "lambda:PublishVersion",
                "lambda:UpdateFunctionCode",
                "lambda:UpdateFunctionConfiguration",
                "lambda:GetFunctionConfiguration",
                "lambda:GetFunction",
                "lambda:DeleteFunction",
                "lambda:ListTags",
                "lambda:TagResource",
                "lambda:UntagResource"
            ],
            "Resource": [
                "arn:aws:lambda:*:*:function:cwsyn-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "lambda:GetLayerVersion",
                "lambda:PublishLayerVersion",
                "lambda:DeleteLayerVersion"
            ],
            "Resource": [
                "arn:aws:lambda:*:*:layer:cwsyn-*",
                "arn:aws:lambda:*:*:layer:Synthetics:*",
                "arn:aws:lambda:*:*:layer:Synthetics_Selenium:*",
                "arn:aws:lambda:*:*:layer:AWS-CW-Synthetics*:*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeVpcs",
```

```
                "ec2:DescribeSubnets",
                "ec2:DescribeSecurityGroups"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "sns:ListTopics"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "sns:CreateTopic",
                "sns:Subscribe",
                "sns:ListSubscriptionsByTopic"
            ],
            "Resource": [
                "arn:*:sns:*:*:Synthetics-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "kms:ListAliases"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "kms:DescribeKey"
            ],
            "Resource": "arn:aws:kms:*:*:key/*"
        },
        {
            "Effect": "Allow",
            "Action": [
```

```
                "kms:Decrypt"
            ],
            "Resource": "arn:aws:kms:*:*:key/*",
            "Condition": {
                "StringLike": {
                    "kms:ViaService": [
                        "s3.*.amazonaws.com"
                    ]
                }
            }
        }
    ]
}
```

CloudWatchSyntheticsReadOnlyAccess

The following is the content of the **CloudWatchSyntheticsReadOnlyAccess** policy.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "synthetics:Describe*",
                "synthetics:Get*",
                "synthetics:List*",
                "lambda:GetFunctionConfiguration"
            ],
            "Resource": "*"
        }
    ]
}
```

Amazon managed (predefined) policies for Amazon CloudWatch RUM

The **AmazonCloudWatchRUMFullAccess** and **AmazonCloudWatchRUMReadOnlyAccess** Amazon managed policies are available for you to assign to users who will manage or use CloudWatch RUM.

AmazonCloudWatchRUMFullAccess

The following are the contents of the **AmazonCloudWatchRUMFullAccess** policy.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "rum:*"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:GetRole",
                "iam:CreateServiceLinkedRole"
            ],
            "Resource": [
                "arn:aws:iam::*:role/aws-service-role/rum.amazonaws.com/
AWSServiceRoleForRealUserMonitoring"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": [
                "arn:aws:iam::*:role/RUM-Monitor*"
            ],
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": [
                        "cognito-identity.amazonaws.com"
                    ]
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:GetMetricData",
                "cloudwatch:GetMetricStatistics",
                "cloudwatch:ListMetrics"
```

```
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:DescribeAlarms"
            ],
            "Resource": "arn:aws:cloudwatch:*:*:alarm:*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "cognito-identity:CreateIdentityPool",
                "cognito-identity:ListIdentityPools",
                "cognito-identity:DescribeIdentityPool",
                "cognito-identity:GetIdentityPoolRoles",
                "cognito-identity:SetIdentityPoolRoles"
            ],
            "Resource": "arn:aws:cognito-identity:*:*:identitypool/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:DeleteLogGroup",
                "logs:PutRetentionPolicy",
                "logs:CreateLogStream"
            ],
            "Resource": "arn:aws:logs:*:*:log-group:*RUMService*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogDelivery",
                "logs:GetLogDelivery",
                "logs:UpdateLogDelivery",
                "logs:DeleteLogDelivery",
                "logs:ListLogDeliveries",
                "logs:DescribeResourcePolicies"
            ],
            "Resource": "*"
        },
        {
```

```
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups"
            ],
            "Resource": "arn:aws:logs:*:*:log-group::log-stream:*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "synthetics:describeCanaries",
                "synthetics:describeCanariesLastRun"
            ],
            "Resource": "arn:aws:synthetics:*:*:canary:*"
        }
    ]
}
```

AmazonCloudWatchRUMReadOnlyAccess

The **AmazonCloudWatchRUMReadOnlyAccess** allows read-only administrative access to CloudWatch RUM.

The following are the contents of the **AmazonCloudWatchRUMReadOnlyAccess** policy.

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "rum:GetAppMonitor",
    "rum:GetAppMonitorData",
    "rum:ListAppMonitors",
    "rum:ListRumMetricsDestinations",
    "rum:BatchGetRumMetricDefinitions",
    "rum:GetResourcePolicy"
   ],
   "Resource": "*"
  }
 ]
}
```

AmazonCloudWatchRUMServiceRolePolicy

You can't attach **AmazonCloudWatchRUMServiceRolePolicy** to your IAM entities. This policy is attached to a service-linked role that allows CloudWatch RUM to publish monitoring data to other relevant Amazon services. For more information about this service linked role, see Using service-linked roles for CloudWatch RUM.

The complete contents of **AmazonCloudWatchRUMServiceRolePolicy** are as follows.

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "xray:PutTraceSegments"
   ],
   "Resource": [
    "*"
   ]
  },
  {
   "Effect": "Allow",
   "Action": "cloudwatch:PutMetricData",
   "Resource": "*",
   "Condition": {
    "StringLike": {
     "cloudwatch:namespace": [
      "RUM/CustomMetrics/*",
      "AWS/RUM"
     ]
    }
   }
  }
 ]
}
```

Amazon managed (predefined) policies for CloudWatch Evidently

The **CloudWatchEvidentlyFullAccess** and **CloudWatchEvidentlyReadOnlyAccess** Amazon managed policies are available for you to assign to users who will manage or use CloudWatch Evidently.

CloudWatchEvidentlyFullAccess

The following are the contents of the **CloudWatchEvidentlyFullAccess** policy.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "evidently:*"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:ListRoles"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:GetRole"
            ],
            "Resource": [
                "arn:aws:iam::*:role/service-role/CloudWatchRUMEvidentlyRole-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketLocation",
                "s3:ListAllMyBuckets"
            ],
            "Resource": "arn:aws:s3:::*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:GetMetricData",
                "cloudwatch:GetMetricStatistics",
                "cloudwatch:DescribeAlarmHistory",
```

```
                "cloudwatch:DescribeAlarmsForMetric",
                "cloudwatch:ListTagsForResource"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:DescribeAlarms",
                "cloudwatch:TagResource",
                "cloudwatch:UnTagResource"
            ],
            "Resource": [
                "arn:aws:cloudwatch:*:*:alarm:*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "cloudtrail:LookupEvents"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:PutMetricAlarm"
            ],
            "Resource": [
                "arn:aws:cloudwatch:*:*:alarm:Evidently-Alarm-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "sns:ListTopics"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
```

```
                "sns:CreateTopic",
                "sns:Subscribe",
                "sns:ListSubscriptionsByTopic"
            ],
            "Resource": [
                "arn:*:sns:*:*:Evidently-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

CloudWatchEvidentlyReadOnlyAccess

The following are the contents of the **CloudWatchEvidentlyReadOnlyAccess** policy.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "evidently:GetExperiment",
                "evidently:GetFeature",
                "evidently:GetLaunch",
                "evidently:GetProject",
                "evidently:GetSegment",
                "evidently:ListExperiments",
                "evidently:ListFeatures",
                "evidently:ListLaunches",
                "evidently:ListProjects",
                "evidently:ListSegments",
                "evidently:ListSegmentReferencs"
            ],
            "Resource": "*"
        }
```

```
    ]
}
```

Amazon managed policy for Amazon Systems Manager Incident Manager

The **AWSCloudWatchAlarms_ActionSSMIncidentsServiceRolePolicy** policy is attached to a service-linked role that allows CloudWatch to start incidents in Amazon Systems Manager Incident Manager on your behalf. For more information, see Service-linked role permissions for CloudWatch alarms Systems Manager Incident Manager actions.

The policy has the following permission:

- ssm-incidents:StartIncident

Customer managed policy examples

In this section, you can find example user policies that grant permissions for various CloudWatch actions. These policies work when you are using the CloudWatch API, Amazon SDKs, or the Amazon CLI.

Examples

- Example 1: Allow user full access to CloudWatch
- Example 2: Allow read-only access to CloudWatch
- Example 3: Stop or terminate an Amazon EC2 instance

Example 1: Allow user full access to CloudWatch

To grant a user full access to CloudWatch, you can use grant them the **CloudWatchFullAccess** managed policy instead of creating a customer-managed policy. The contents of the **CloudWatchFullAccess** are listed in CloudWatchFullAccess.

Example 2: Allow read-only access to CloudWatch

The following policy allows a user read-only access to CloudWatch and view Amazon EC2 Auto Scaling actions, CloudWatch metrics, CloudWatch Logs data, and alarm-related Amazon SNS data.

```
{
  "Version": "2012-10-17",
  "Statement": [
```

```
    {
      "Action": [
        "autoscaling:Describe*",
        "cloudwatch:Describe*",
        "cloudwatch:Get*",
        "cloudwatch:List*",
        "logs:Get*",
        "logs:Describe*",
        "logs:StartQuery",
        "logs:StopQuery",
        "logs:TestMetricFilter",
        "logs:FilterLogEvents",
        "logs:StartLiveTail",
        "logs:StopLiveTail",
        "sns:Get*",
        "sns:List*"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

Example 3: Stop or terminate an Amazon EC2 instance

The following policy allows an CloudWatch alarm action to stop or terminate an EC2 instance. In the sample below, the GetMetricData, ListMetrics, and DescribeAlarms actions are optional. It is recommended that you include these actions to ensure that you have correctly stopped or terminated the instance.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "cloudwatch:PutMetricAlarm",
        "cloudwatch:GetMetricData",
        "cloudwatch:ListMetrics",
        "cloudwatch:DescribeAlarms"
      ],
      "Resource": [
        "*"
      ],
```

```
      "Effect": "Allow"
    },
    {
      "Action": [
        "ec2:DescribeInstanceStatus",
        "ec2:DescribeInstances",
        "ec2:StopInstances",
        "ec2:TerminateInstances"
      ],
      "Resource": [
        "*"
      ],
      "Effect": "Allow"
    }
  ]
}
```

CloudWatch updates to Amazon managed policies

View details about updates to Amazon managed policies for CloudWatch since this service began tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the CloudWatch Document history page.

Change	Description	Date
AmazonCloudWatchRUMReadOnlyAccess – Updated policy	CloudWatch added a permission to the **AmazonCloudWatchRUMReadOnlyAccess** policy. The `rum:GetResourcePolicy` permission was added so that CloudWatch RUM can view the resource policy attached to the RUM application monitor.	April 28, 2025
AIOpsConsoleAdminPolicy – New policy	CloudWatch created a new policy named **AIOpsConsoleAdminPolicy**.	December 3, 2024

Change	Description	Date
	This policy grants users full administrative access for managing Amazon Q Developer operational investigations, including the management of trusted identity propagation, and the management of IAM Identity Center and organizational access.	
AIOpOperatorAccess – New policy	CloudWatch created a new policy named **AIOpsOperatorAccess**. This policy grants users access to Amazon Q Developer operational investigations actions, and to additional Amazon actions that are necessary for accessing investigation events.	December 3, 2024
AIOpsReadOnlyAccess – New policy	CloudWatch created a new policy named **AIOpsReadOnlyAccess**. This policy grants a user read-only permissions for Amazon AI Operations and other related services.	December 3, 2024

Change	Description	Date
AIOpsAssistantPolicy – New policy	CloudWatch created a new policy named **AIOpsAssistantPolicy**. You don't assign this policy to a user. You assign this policy to the Amazon AI Operations assistant to enable Amazon Q operational investigations to analyze your Amazon resources during the investigation of operational events.	December 3, 2024
CloudWatchFullAccessV2 – Updates to existing policies	CloudWatch updated both **CloudWatchFullAccessV2** and **CloudWatchFullAccess**. Permissions for Amazon OpenSearch Service were added to to enable CloudWatch Logs integration with OpenSearch Service for some features.	December 1, 2024

Change	Description	Date
[CloudWatchNetworkFlowMonitorServiceRolePolicy](#) – New policy	CloudWatch added a new policy **CloudWatchNetworkFlowMonitorServiceRolePolicy**. The **CloudWatchNetworkFlowMonitorServiceRolePolicy** grants permissions for Network Flow Monitor to publish metrics to CloudWatch. It also allows the service to use Amazon Organizations to get information for multi-account scenarios.	December 1, 2024
[CloudWatchNetworkFlowMonitorTopologyServiceRolePolicy](#) – New policy	CloudWatch added a new policy **CloudWatchNetworkFlowMonitorTopologyServiceRolePolicy**. The **CloudWatchNetworkFlowMonitorTopologyServiceRolePolicy** grants permissions for Network Flow Monitor to generate topology snapshots of resources used in your account.	December 1, 2024

Change	Description	Date
[CloudWatchNetworkFlowMonitorAgentPublishPolicy]() – New policy	CloudWatch added a new policy **CloudWatchNetworkFlowMonitorAgentPublishPolicy**. The **CloudWatchNetworkFlowMonitorAgentPublishPolicy** grants permissions for resources, such as Amazon EC2 and Amazon EKS instances, to send telemetry reports (metrics) to a Network Flow Monitor endpoint.	December 1, 2024

Change	Description	Date
[CloudWatchSyntheticsFullAccess]() – Update to an existing policy	CloudWatch updated the policy named **CloudWatchSyntheticsFullAccess**. The following CloudWatch Logs actions have been added to allow CloudWatch Synthetics to get and use canary log data in Lambda log groups. The `lambda:GetFunction` permission has also been added to allow Synthetics to get information about a specific function. • `logs:GetLogRecord` – Required to expand a log entry in CloudWatch Logs Insights. (Required to view logs in the Synthetics console.) • `logs:DescribeLogStreams` – Required to list all log streams on a log group. • `logs:StartQuery` – Required to start a Logs Insights query (required in the Synthetics console to view logs). • `logs:GetLogEvents` – Required to list log events from the specified log stream. Used for querying	November 20, 2024

Change	Description	Date
	a specific log stream on a log group. • `logs:FilterLogEvents` – Required to view logs in a canary's log group. • `logs:GetLogGroupFields` – Required for running a Logs Insights query in the console. (The Synthetics console links to the Logs Insights query. Without this permission, Logs Insights queries on a log group fail.) Additionally, Lambda layer version actions now apply to all CloudWatch Synthetics layer ARNs.	

Change	Description	Date
[CloudWatchInternetMonitorReadOnlyAccess]() – New **CloudWatchInternetMonitorReadOnlyAccess**. This policy grants read only access to resources and actions available in the CloudWatch console for Internet Monitor. The scope of this policy includes `internetmonitor:` so that users can use read-only Internet Monitor actions and resources. It includes some `cloudwatch:` policies to retrieve information on CloudWatch metrics. It includes some `logs:` policies to manage log queries.	November 14, 2024	

Change	Description	Date
CloudWatchInternetMonitorFullAccess – New policy	CloudWatch created a new policy named **CloudWatchInternetMonitorFullAccess**. This policy grants full access to resources and actions available in the CloudWatch console for Internet Monitor. The scope of this policy includes `internetmonitor:` so that users can use Internet Monitor actions and resources. It includes some `cloudwatch:` policies to retrieve information on CloudWatch alarms and metrics. It includes some `logs:` policies to manage log queries. It includes some `ec2:`, `cloudfront:` , `elasticloadbalancing:` , and `workspaces:` policies to work with resources that you add to monitors so that Internet Monitor can create a traffic profile for your application. It contains some `iam:` policies to manage IAM roles.	October 23, 2024

Change	Description	Date
[CloudWatchLambdaApplicationSignalsExecutionRolePolicy]() – New **CloudWatchLambdaApplicationSignalsExecutionRolePolicy**. This policy is used when CloudWatch Application Signals is enabled for Lambda workloads. It enables write access to X-Ray and the log group used by CloudWatch Application Signals.	October 16, 2024	
[CloudWatchSyntheticsFullAccess]() – Update to an existing policy	CloudWatch updated the policy named **CloudWatchSyntheticsFullAccess**. The `lambda:ListTags` , `lambda:TagResource` , and `lambda:UntagResource` permissions were added so that when you apply or change tags on a canary, you can choose to have Synthetics also apply those same tags or changes to the Lambda function that the canary uses.	October 11, 2024

Change	Description	Date
CloudWatchApplicationSignalsReadOnlyAccess – New policy	CloudWatch created a new policy named **CloudWatchApplicationSignalsReadOnlyAccess**. This policy grants read only access to resources and actions available in the CloudWatch console for Application Signals. The scope of this policy includes `application-signals:` policies so that users can use read only actions and resources available in the CloudWatch console under Application Signals. It contains an `iam:` policy to manage IAM roles. It includes some `logs:` policies to manage log queries and filters. It includes `cloudwatch:` policies to retrieve information on CloudWatch alarms and metrics. It includes some `synthetics:` policies to retrive information about synthetics canaries. It includes `rum:` policies to manage RUM clients and jobs. It contains an `xray:` policy to obtain trace summaries.	June 7, 2024

Change	Description	Date
CloudWatchApplicationSignalsFullAccess – New policy	CloudWatch created a new policy named **CloudWatchApplicationSignalsFullAccess**. This policy grants full access to resources and actions available in the CloudWatch console for Application Signals. The scope of this policy includes `application-signals:` so that users can use Application Signals actions and resources. It includes some `cloudwatch:` policies to retrieve information on CloudWatch alarms and metrics. It includes some `logs:` policies to manage log queries. It includes some `synthetics:` policies to write and retrieve information about synthetics canaries. It includes `rum:` policies to manage RUM clients and jobs. It contains an `xray:` policy to obtain trace summaries. It includes some `cloudwatch:` policies to manage CloudWatch alarms. It contains some `iam:` policies to manage IAM roles. It includes some `sns:` policies to manage Amazon Simple	June 7, 2024

Change	Description	Date
	Notification Service notificat ions.	
[CloudWatchFullAccessV2](#) – Update to an existing policy	CloudWatch updated the policy named **CloudWatchFullAccessV2**. The scope of the `CloudWatchFullAccessPermissions` policy was updated to add `application-signals:*` so that users can use CloudWatch Application Signals to view, investigate, and diagnose issues with the health of their services.	May 20, 2024

Change	Description	Date
CloudWatchReadOnlyAccess – Update to an existing policy	CloudWatch updated the policy named **CloudWatchReadOnlyAccess**. The scope of the `CloudWatchReadOnlyAccessPermissions` policy was updated to add `application-signals:BatchGet*` , `application-signals:List*` , and `application-signals:Get*` so that users can use CloudWatch Application Signals to view, investigate, and diagnose issues with the health of their services. The scope of `CloudWatchReadOnlyGetRolePermissions` was updated to add the `iam:GetRole` action so that users can check if CloudWatch Application Signals is set up.	May 20, 2024

Change	Description	Date
CloudWatchApplicationSignalsServiceRolePolicy – Update to an existing policy	CloudWatch updated the policy named **CloudWatchApplicationSignalsServiceRolePolicy**. The scoping of the `logs:StartQuery` and `logs:GetQueryResults` permissions was changed to add the `arn:aws:logs:*:*:log-group:/aws/appsignals/*:*` and `arn:aws:logs:*:*:log-group:/aws/application-signals/data:*` ARNs to enable Application Signals on more architectures.	April 18, 2024
CloudWatchApplicationSignalsServiceRolePolicy – Update to an existing policy	CloudWatch changed the scope of a permission in **CloudWatchApplicationSignalsServiceRolePolicy**. The scope of the `cloudwatch:GetMetricData` permission was changed to * so that Application Signals can retrieve metrics from sources in linked accounts.	April 08, 2024

Change	Description	Date
CloudWatchFullAccessV2 – Update to an existing policy	CloudWatch added permissions to **CloudWatchFullAccessV2**. Existing permissions for CloudWatch Synthetics, X-Ray, and CloudWatch RUM actions and new permissions for CloudWatch Application Signals were added so that users with this policy can manage CloudWatch Application Signals. The permission to create the CloudWatch Application Signals service-linked role was added to allow CloudWatch Application Signals to discover telemetry data in logs, metrics, traces, and tags.	December 5, 2023

Change	Description	Date
CloudWatchReadOnlyAccess – Update to an existing policy	CloudWatch added permissions to **CloudWatchReadOnlyAccess**. Existing read-only permissions for CloudWatch Synthetics, X-Ray, and CloudWatch RUM actions and new read-only permissions for CloudWatch Application Signals were added so that users with this policy can triage and dignose their service health issues as reported by CloudWatch Application Signals. The `cloudwatch:GenerateQuery` permission was added so that users with this policy can generate a CloudWatch Metrics Insights query string from a natural language prompt.	December 5, 2023
CloudWatchReadOnlyAccess – Update to an existing policy.	CloudWatch added a permission to **CloudWatchReadOnlyAccess**. The `cloudwatch:GenerateQuery` permission was added, so that users with this policy can generate a CloudWatch Metrics Insights query string from a natural language prompt.	December 01, 2023

Change	Description	Date
[CloudWatchApplicationSignalsServiceRolePolicy](#) – New policy	CloudWatch added a new policy **CloudWatchApplicationSignalsServiceRolePolicy**. The **CloudWatchApplicationSignalsServiceRolePolicy** grants an upcoming feature permissions to collect CloudWatch Logs data, X-Ray trace data, CloudWatch metrics data, and tagging data.	November 9, 2023
[AWSServiceRoleForCloudWatchMetrics_DbPerfInsightsServiceRolePolicy](#) – New policy	CloudWatch added a new policy **AWSServiceRoleForCloudWatchMetrics_DbPerfInsightsServiceRolePolicy**. The **AWSServiceRoleForCloudWatchMetrics_DbPerfInsightsServiceRolePolicy** grants permission to CloudWatch to fetch Performance Insights metrics from databases on your behalf.	September 20, 2023

Change	Description	Date
CloudWatchReadOnlyAccess – Update to an existing policy	CloudWatch added a permission to **CloudWatchReadOnlyAccess**. The `application-autoscaling:DescribeScalingPolicies` permission was added so that users with this policy can access information about Application Auto Scaling policies.	September 14, 2023
CloudWatchFullAccessV2 – New policy	CloudWatch added a new policy **CloudWatchFullAccessV2**. The **CloudWatchFullAccessV2** grants full access to CloudWatch actions and resources while better scoping the permissions granted to other services such as Amazon SNS and Amazon EC2 Auto Scaling. For more information, see CloudWatchFullAccessV2.	August 1, 2023

Change	Description	Date
AWSServiceRoleForInternetMonitor – Update to an existing policy	Amazon CloudWatch Internet Monitor added new permissions to monitor Network Load Balancer resources. The `elasticloadbalancing:DescribeLoadBalancers` and `ec2:DescribeNetworkInterfaces` permissions are required so that Internet Monitor can monitor customers' Network Load Balancer traffic by analyzing flow logs for NLB resources. For more information, see Using Internet Monitor.	July 15, 2023
CloudWatchReadOnlyAccess – Update to an existing policy	CloudWatch added permissions to **CloudWatchReadOnlyAccess**. The `logs:StartLiveTail` and `logs:StopLiveTail` permissions were added so that users with this policy can use the console to start and stop CloudWatch Logs live tail sessions. For more information, see Use live tail to view logs in near real time.	June 6, 2023

Change	Description	Date
CloudWatchCrossAccountSharingConfiguration – New policy	CloudWatch added a new policy to enable you to manage CloudWatch cross-account observability links that share CloudWatch metrics. For more information, see CloudWatch cross-account observability.	November 27, 2022
OAMFullAccess – New policy	CloudWatch added a new policy to enable you to fully manage CloudWatch cross-account observability links and sinks. For more information, see CloudWatch cross-account observability.	November 27, 2022
OAMReadOnlyAccess – New policy	CloudWatch added a new policy to enable you to view information about CloudWatch cross-account observability links and sinks. For more information, see CloudWatch cross-account observability.	November 27, 2022

Change	Description	Date
CloudWatchFullAccess – Update to an existing policy	CloudWatch added permissions to **CloudWatchFullAccess**. The `oam:ListSinks` and `oam:ListAttachedLinks` permissions were added so that users with this policy can use the console to view data shared from source accounts in CloudWatch cross-account observability.	November 27, 2022
CloudWatchReadOnlyAccess – Update to an existing policy	CloudWatch added permissions to **CloudWatchReadOnlyAccess**. The `oam:ListSinks` and `oam:ListAttachedLinks` permissions were added so that users with this policy can use the console to view data shared from source accounts in CloudWatch cross-account observability.	November 27, 2022

Change	Description	Date
AmazonCloudWatchRUMServiceRolePolicy – Update to an existing policy	CloudWatch RUM updated a condition key in **AmazonCloudWatchRUMServiceRolePolicy**. The `"Condition": { "StringEquals": { "cloudwatch:namespace": "AWS/RUM" } }` condition key was changed to the following so that CloudWatch RUM can send custom metrics to custom metric namespaces. `"Condition": { "StringLike": { "cloudwatch:namespace": [ "RUM/CustomMetrics/*", "AWS/RUM" ] } }`	February 2, 2023

Change	Description	Date
AmazonCloudWatchRUMReadOnlyAccess – Updated policy	CloudWatch added permissions the **AmazonCloudWatchRUMReadOnlyAccess** policy. The `rum:ListRumMetricsDestinations` and `rum:BatchGetRumMetricsDefinitions` permissions were added so that CloudWatch RUM can send extended metrics to CloudWatch and Evidently.	October 27, 2022
AmazonCloudWatchRUMServiceRolePolicy – Update to an existing policy	CloudWatch RUM added permissions to **AmazonCloudWatchRUMServiceRolePolicy**. The `cloudwatch:PutMetricData` permission was added so that CloudWatch RUM can send extended metrics to CloudWatch.	October 26, 2022

Change	Description	Date
CloudWatchEvidentlyReadOnlyAccess – Update to an existing policy	CloudWatch Evidently added permissions to **CloudWatchEvidentlyReadOnlyAccess**. The `evidently:GetSegment`, `evidently:ListSegments`, and `evidently:ListSegmentReferences` permissions were added so that users with this policy can see Evidently audience segments that have been created.	August 12, 2022
CloudWatchSyntheticsFullAccess – Update to an existing policy	CloudWatch Synthetics added permissions to **CloudWatchSyntheticsFullAccess**. The `lambda:DeleteFunction` and `lambda:DeleteLayerVersion` permissions were added so that CloudWatch Synthetics can delete related resources when a canary is deleted. The `iam:ListAttachedRolePolicies` was added so that customers can view the policies that are attached to a canary's IAM role.	May 6, 2022

Change	Description	Date
AmazonCloudWatchRUMFullAccess – New policy	CloudWatch added a new policy to enable full management of CloudWatch RUM. CloudWatch RUM allows you to perform real user monitoring of your web application. For more information, see CloudWatch RUM.	November 29, 2021
AmazonCloudWatchRUMReadOnlyAccess – New policy	CloudWatch added a new policy to enable read-only access to CloudWatch RUM. CloudWatch RUM allows you to perform real user monitoring of your web application. For more information, see CloudWatch RUM.	November 29, 2021
CloudWatchEvidentlyFullAccess – New policy	CloudWatch added a new policy to enable full management of CloudWatch Evidently. CloudWatch Evidently allows you to perform A/B experiments of your web applications, and to roll them out gradually. For more information, see Perform launches and A/B experiments with CloudWatch Evidently.	November 29, 2021

Change	Description	Date
CloudWatchEvidentlyReadOnlyAccess – New policy	CloudWatch added a new policy to enable read-only access to CloudWatch Evidently. CloudWatch Evidently allows you to perform A/B experiments of your web applications, and to roll them out gradually. For more information, see Perform launches and A/B experiments with CloudWatch Evidently.	November 29, 2021
AWSServiceRoleForCloudWatchRUM – New managed policy	CloudWatch added a policy for a new service-linked role to allow CloudWatch RUM to pubish monitoring data to other relevant Amazon services.	November 29, 2021

Change	Description	Date
CloudWatchSyntheticsFullAccess – Update to an existing policy	CloudWatch Synthetics added permissions to **CloudWatchSyntheticsFullAccess**, and also changed the scope of one permission. The `kms:ListAliases` permission was added so that users can list available Amazon KMS keys that can be used to encrypt canary artifacts. The `kms:DescribeKey` permission was added so that users can see the details of keys that will be used to encrypt for canary artifacts. And the `kms:Decrypt` permission was added to enable users to decrypt canary artifacts. This decryption ability is limited to use on resources within Amazon S3 buckets. The `Resource` scope of the `s3:GetBucketLocation` permission was changed from `*` to `arn:aws:s3:::*` .	September 29, 2021

Change	Description	Date
CloudWatchSyntheticsFullAccess – Update to an existing policy	CloudWatch Synthetics added a permission to **CloudWatchSyntheticsFullAccess**. The `lambda:UpdateFunctionCode` permission was added so that users with this policy can change the runtime version of canaries.	July 20, 2021
AWSCloudWatchAlarms_ActionSSMIncidentsServiceRolePolicy – New managed policy	CloudWatch added a new managed IAM policy to allow CloudWatch to create incidents in Amazon Systems Manager Incident Manager.	May 10, 2021
CloudWatchAutomaticDashboardsAccess – Update to an existing policy	CloudWatch added a permission to the **CloudWatchAutomaticDashboardsAccess** managed policy. The `synthetics:DescribeCanariesLastRun` permission was added to this policy to enable cross-account dashboard users to see details about CloudWatch Synthetics canary runs.	April 20, 2021
CloudWatch started tracking changes	CloudWatch started tracking changes for its Amazon managed policies.	April 14, 2021

Using condition keys to limit access to CloudWatch namespaces

Use IAM condition keys to limit users to publishing metrics only in the CloudWatch namespaces that you specify. This section provides examples that describe how to allow and exclude users from publishing metrics in a namespace.

Allowing publishing in one namespace only

The following policy limits the user to publishing metrics only in the namespace named `MyCustomNamespace`.

```
{
    "Version": "2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Resource": "*",
        "Action": "cloudwatch:PutMetricData",
        "Condition": {
            "StringEquals": {
                "cloudwatch:namespace": "MyCustomNamespace"
            }
        }
    }
}
```

Excluding publishing from a namespace

The following policy allows the user to publish metrics in any namespace except for `CustomNamespace2`.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Resource": "*",
            "Action": "cloudwatch:PutMetricData"
        },
        {
            "Effect": "Deny",
            "Resource": "*",
```

```
            "Action": "cloudwatch:PutMetricData",
            "Condition": {
                "StringEquals": {
                    "cloudwatch:namespace": "CustomNamespace2"
                }
            }
        }
    ]
}
```

Using condition keys to limit Contributor Insights users' access to log groups

To create a rule in Contributor Insights and see its results, a user must have the `cloudwatch:PutInsightRule` permission. By default, a user with this permission can create a Contributor Insights rule that evaluates any log group in CloudWatch Logs and then see the results. The results can contain contributor data for those log groups.

You can create IAM policies with condition keys to grant users the permission to write Contributor Insights rules for some log groups while preventing them from writing rules for and seeing this data from other log groups.

For more information about the `Condition` element in IAM policies, see [IAM JSON policy elements: Condition](#).

Allow access to write rules and view results for only certain log groups

The following policy allows the user access to write rules and view results for the log group named `AllowedLogGroup` and all log groups that have names that start with `AllowedWildCard`. It does not grant access to write rules or view rule results for any other log groups.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowCertainLogGroups",
            "Effect": "Allow",
            "Action": "cloudwatch:PutInsightRule",
            "Resource": "arn:aws:cloudwatch:*:*:insight-rule/*",
            "Condition": {
```

```
                "ForAllValues:StringEqualsIgnoreCase": {
                    "cloudwatch:requestInsightRuleLogGroups": [
                        "AllowedLogGroup",
                        "AllowedWildcard*"
                    ]
                }
            }
        }
    ]
}
```

Deny writing rules for specific log groups but allow writing rules for all other log groups

The following policy explicitly denies the user access to write rules and view rule results for the log group named `ExplicitlyDeniedLogGroup`, but allows writing rules and viewing rule results for all other log groups.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowInsightRulesOnLogGroupsByDefault",
            "Effect": "Allow",
            "Action": "cloudwatch:PutInsightRule",
            "Resource": "arn:aws:cloudwatch:*:*:insight-rule/*"

        },
        {
            "Sid": "ExplicitDenySomeLogGroups",
            "Effect": "Deny",
            "Action": "cloudwatch:PutInsightRule",
            "Resource": "arn:aws:cloudwatch:*:*:insight-rule/*",
            "Condition": {
                "ForAllValues:StringEqualsIgnoreCase": {
                    "cloudwatch:requestInsightRuleLogGroups": [
                        "/test/alpine/ExplicitlyDeniedLogGroup"
                    ]
                }
            }
        }
    ]
}
```

Using condition keys to limit alarm actions

When CloudWatch alarms change state, they can perform different actions such as stopping and terminating EC2 instances and performing Systems Manager actions. These actions can be initiated when the alarm changes to any state, including ALARM, OK, or INSUFFICIENT_DATA.

Use the `cloudwatch:AlarmActions` condition key to allow a user to create alarms that can only perform the actions you specify when the alarm state changes. For example, you can allow a user to create alarms that can only perform actions which are not EC2 actions.

Allow a user to create alarms that can only send Amazon SNS notifications or perform Systems Manager actions

The following policy limits the user to creating alarms that can only send Amazon SNS notifications and perform Systems Manager actions. The user can't create alarms that perform EC2 actions.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CreateAlarmsThatCanPerformOnlySNSandSSMActions",
            "Effect": "Allow",
            "Action": "cloudwatch:PutMetricAlarm",
            "Resource": "*",
            "Condition": {
                "ForAllValues:StringLike": {
                    "cloudwatch:AlarmActions": [
                        "arn:aws:sns:*",
                        "arn:aws:ssm:*"
                    ]
                }
            }
        }
    ]
}
```

Using service-linked roles for CloudWatch

Amazon CloudWatch uses Amazon Identity and Access Management (IAM) service-linked roles. A service-linked role is a unique type of IAM role that is linked directly to CloudWatch. Service-linked roles are predefined by CloudWatch and include all the permissions that the service requires to call other Amazon services on your behalf.

One service-linked role in CloudWatch makes setting up CloudWatch alarms that can terminate, stop, or reboot an Amazon EC2 instance without requiring you to manually add the necessary permissions. Another service-linked role enables a monitoring account to access CloudWatch data from other accounts that you specify, to build cross-account cross-Region dashboards.

CloudWatch defines the permissions of these service-linked roles, and unless defined otherwise, only CloudWatch can assume the role. The defined permissions include the trust policy and the permissions policy, and that permissions policy cannot be attached to any other IAM entity.

You can delete the roles only after first deleting their related resources. This restriction protects your CloudWatch resources because you can't inadvertently remove permissions to access the resources.

For information about other services that support service-linked roles, see Amazon Services That Work with IAM and look for the services that have **Yes** in the **Service-Linked Role** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

Service-linked role permissions for CloudWatch alarms EC2 actions

CloudWatch uses the service-linked role named **AWSServiceRoleForCloudWatchEvents** – CloudWatch uses this service-linked role to perform Amazon EC2 alarm actions.

The AWSServiceRoleForCloudWatchEvents service-linked role trusts the CloudWatch Events service to assume the role. CloudWatch Events invokes the terminate, stop, or reboot instance actions when called upon by the alarm.

The AWSServiceRoleForCloudWatchEvents service-linked role permissions policy allows CloudWatch Events to complete the following actions on Amazon EC2 instances:

- `ec2:StopInstances`
- `ec2:TerminateInstances`
- `ec2:RecoverInstances`
- `ec2:DescribeInstanceRecoveryAttribute`
- `ec2:DescribeInstances`
- `ec2:DescribeInstanceStatus`

The **AWSServiceRoleForCloudWatchCrossAccount** service-linked role permissions policy allows CloudWatch to complete the following actions:

- `sts:AssumeRole`

Service-linked role permissions for CloudWatch telemetry config

CloudWatch observability admin creates and uses a service-linked role named **AWSServiceRoleForObservabilityAdmin –** CloudWatch uses this service-linked role to support resource and telemetry config discovery for Amazon Organizations. The role is created in all member accounts of the Organization.

The **AWSServiceRoleForObservabilityAdmin** service-linked role trusts Observability Admin to assume the role. Observability Admin manages Amazon Config Service Linked Configuration Recorders and Service Linked Configuration Aggregator in your Organizations accounts.

The **AWSServiceRoleForObservabilityAdmin** service-linked role has a policy, called AWSObservabilityAdminServiceRolePolicy, attached and this policy grants permission to CloudWatch Observability Admin to complete the following actions:

- `organizations:ListAccounts`
- `organizations:ListAccountsForParent`
- `organizations:ListChildren`
- `organizations:ListParents`
- `organizations:DescribeOrganization`
- `organizations:DescribeOrganizationalUnit`
- `organizations:EnableAWSServiceAccess`
- `organizations:ListDelegatedAdministrators`
- `config:PutServiceLinkedConfigurationRecorder`
- `config:DeleteServiceLinkedConfigurationRecorder`
- `config:PutConfigurationAggregator`
- `config:DeleteConfigurationAggregator`
- `config:SelectAggregateResourceConfig`
- `iam:CreateServiceLinkedRole`
- `iam:PassRole`

The complete contents of the AWSObservabilityAdminServiceRolePolicy policy are as follows:

```
 {
 "Version": "2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "organizations:ListAccounts",
    "organizations:ListAccountsForParent",
    "organizations:ListChildren",
    "organizations:ListParents",
    "organizations:DescribeOrganization",
    "organizations:DescribeOrganizationalUnit"
   ],
   "Resource": "*"
  },
  {
   "Effect": "Allow",
   "Action": [
    "config:PutServiceLinkedConfigurationRecorder",
    "config:DeleteServiceLinkedConfigurationRecorder"
   ],
   "Resource": [
    "arn:aws:config:*:*:configuration-recorder/
AWSConfigurationRecorderForObservabilityAdmin/*"
   ]
  },
  {
   "Effect": "Allow",
   "Action": [
    "config:PutConfigurationAggregator",
    "config:DeleteConfigurationAggregator",
    "config:SelectAggregateResourceConfig"
   ],
   "Resource": [
    "arn:aws:config:*:*:config-aggregator/aws-service-config-aggregator/
observabilityadmin.amazonaws.com/*"
   ]
  },
  {
   "Effect": "Allow",
   "Action": [
```

```
      "iam:CreateServiceLinkedRole"
    ],
    "Resource": [
      "arn:aws:iam::*:role/aws-service-role/config.amazonaws.com/AWSServiceRoleForConfig"
    ],
    "Condition": {
      "StringEquals": {
        "iam:AWSServiceName": [
          "config.amazonaws.com"
        ]
      }
    }
  },
  {
    "Effect": "Allow",
    "Action": [
      "iam:PassRole"
    ],
    "Resource": [
      "arn:aws:iam::*:role/aws-service-role/config.amazonaws.com/AWSServiceRoleForConfig"
    ],
    "Condition": {
      "StringEquals": {
        "iam:PassedToService": [
          "config.amazonaws.com"
        ]
      }
    }
  },
  {
    "Effect": "Allow",
    "Action": [
      "organizations:EnableAWSServiceAccess"
    ],
    "Resource": "*",
    "Condition": {
      "StringEquals": {
        "organizations:ServicePrincipal": [
          "config.amazonaws.com"
        ]
      }
    }
  },
  {
```

```
    "Effect": "Allow",
    "Action": [
     "organizations:ListDelegatedAdministrators"
    ],
    "Resource": "*",
    "Condition": {
     "StringEquals": {
      "organizations:ServicePrincipal": [
       "observabilityadmin.amazonaws.com",
       "config.amazonaws.com"
      ]
     }
    }
   }
  ]
 }
```

Service-linked role permissions for CloudWatch Application Signals

CloudWatch Application Signals uses the service-linked role named **AWSServiceRoleForCloudWatchApplicationSignals** – CloudWatch uses this service-linked role to collect CloudWatch Logs data, X-Ray trace data, CloudWatch metrics data, and tagging data from applications that you have enabled for CloudWatch Application Signals.

The **AWSServiceRoleForCloudWatchApplicationSignals** service-linked role trusts CloudWatch Application Signals to assume the role. Application Signals collects the logs, traces, metrics, and tags data from your account.

The **AWSServiceRoleForCloudWatchApplicationSignals** has an IAM policy attached, and this policy is named **CloudWatchApplicationSignalsServiceRolePolicy**. This policy grants permission to CloudWatch Application Signals to collect monitoring and tagging data from other relevant Amazon services. It includes permissions for Application Signals to complete the following actions:

- `xray` – Retrieve X-Ray traces.
- `logs` – Retrieve the current CloudWatch logs information.
- `cloudwatch` – Retrieve the current CloudWatch metric information.
- `tags` – Retrieve the current tags.
- `application-signals` – Retrieve information on SLOs and their associated time exclusion windows.

- `autoscaling` – Retrieve application tags from Amazon EC2 Autoscaling group.

The complete contents of **CloudWatchApplicationSignalsServiceRolePolicy** are as follows:

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Sid": "XRayPermission",
   "Effect": "Allow",
   "Action": [
    "xray:GetServiceGraph"
   ],
   "Resource": [
    "*"
   ],
   "Condition": {
    "StringEquals": {
     "aws:ResourceAccount": "${aws:PrincipalAccount}"
    }
   }
  },
  {
   "Sid": "CWLogsPermission",
   "Effect": "Allow",
   "Action": [
    "logs:StartQuery",
    "logs:GetQueryResults"
   ],
   "Resource": [
    "arn:aws:logs:*:*:log-group:/aws/appsignals/*:*",
    "arn:aws:logs:*:*:log-group:/aws/application-signals/data:*"
   ],
   "Condition": {
    "StringEquals": {
     "aws:ResourceAccount": "${aws:PrincipalAccount}"
    }
   }
  },
  {
   "Sid": "CWListMetricsPermission",
   "Effect": "Allow",
   "Action": [
```

```
      "cloudwatch:ListMetrics"
    ],
    "Resource": [
      "*"
    ],
    "Condition": {
      "StringEquals": {
        "aws:ResourceAccount": "${aws:PrincipalAccount}"
      }
    }
  },
  {
    "Sid": "CWGetMetricDataPermission",
    "Effect": "Allow",
    "Action": [
      "cloudwatch:GetMetricData"
    ],
    "Resource": [
      "*"
    ]
  },
  {
    "Sid": "TagsPermission",
    "Effect": "Allow",
    "Action": [
      "tag:GetResources"
    ],
    "Resource": [
      "*"
    ],
    "Condition": {
      "StringEquals": {
        "aws:ResourceAccount": "${aws:PrincipalAccount}"
      }
    }
  },
  {
    "Sid": "ApplicationSignalsPermission",
    "Effect": "Allow",
    "Action": [
      "application-signals:ListServiceLevelObjectiveExclusionWindows",
         "application-signals:GetServiceLevelObjective"
    ],
    "Resource": [
```

```
      "*"
    ],
    "Condition": {
     "StringEquals": {
      "aws:ResourceAccount": "${aws:PrincipalAccount}"
     }
    }
   },
   {
    "Sid": "EC2AutoScalingPermission",
    "Effect": "Allow",
    "Action": [
     "autoscaling:DescribeAutoScalingGroups"
    ],
    "Resource": [
     "*"
    ],
    "Condition": {
     "StringEquals": {
      "aws:ResourceAccount": "${aws:PrincipalAccount}"
     }
    }
   }
  ]
}
```

Service-linked role permissions for CloudWatch alarms Systems Manager OpsCenter actions

CloudWatch uses the service-linked role named **AWSServiceRoleForCloudWatchAlarms_ActionSSM** – CloudWatch uses this service-linked role to perform Systems Manager OpsCenter actions when a CloudWatch alarm goes into ALARM state.

The AWSServiceRoleForCloudWatchAlarms_ActionSSM service-linked role trusts the CloudWatch service to assume the role. CloudWatch alarms invoke the Systems Manager OpsCenter actions when called upon by the alarm.

The **AWSServiceRoleForCloudWatchAlarms_ActionSSM** service-linked role permissions policy allows Systems Manager to complete the following actions:

- `ssm:CreateOpsItem`

Service-linked role permissions for CloudWatch alarms Systems Manager Incident Manager actions

CloudWatch uses the service-linked role named **AWSServiceRoleForCloudWatchAlarms_ActionSSMIncidents** – CloudWatch uses this service-linked role to start Incident Manager incidents when a CloudWatch alarm goes into ALARM state.

The **AWSServiceRoleForCloudWatchAlarms_ActionSSMIncidents** service-linked role trusts the CloudWatch service to assume the role. CloudWatch alarms invoke the Systems Manager Incident Manager action when called upon by the alarm.

The **AWSServiceRoleForCloudWatchAlarms_ActionSSMIncidents** service-linked role permissions policy allows Systems Manager to complete the following actions:

- `ssm-incidents:StartIncident`

Service-linked role permissions for CloudWatch cross-account cross-Region

CloudWatch uses the service-linked role named **AWSServiceRoleForCloudWatchCrossAccount** – CloudWatch uses this role to access CloudWatch data in other Amazon accounts that you specify. The SLR only provides the assume role permission to allow the CloudWatch service to assume the role in the sharing account. It is the sharing role that provides access to data.

The **AWSServiceRoleForCloudWatchCrossAccount** service-linked role permissions policy allows CloudWatch to complete the following actions:

- `sts:AssumeRole`

The **AWSServiceRoleForCloudWatchCrossAccount** service-linked role trusts the CloudWatch service to assume the role.

Service-linked role permissions for CloudWatch database Performance Insights

CloudWatch uses the service-linked role named **AWSServiceRoleForCloudWatchMetrics_DbPerfInsights**. – CloudWatch uses this role to retrieve Performance Insights metrics for creating alarms and snapshotting.

The **AWSServiceRoleForCloudWatchMetrics_DbPerfInsights** service-linked role has the `AWSServiceRoleForCloudWatchMetrics_DbPerfInsightsServiceRolePolicy` IAM policy attached. The contents of that policy are as follows:

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "pi:GetResourceMetrics"
   ],
   "Resource": "*",
   "Condition": {
    "StringEquals": {
     "aws:ResourceAccount": "${aws:PrincipalAccount}"
    }
   }
  }
 ]
}
```

The **AWSServiceRoleForCloudWatchMetrics_DbPerfInsights** service-linked role trusts the CloudWatch service to assume the role.

Creating a service-linked role for CloudWatch

You do not need to manually create any of these service-linked roles. The first time you create an alarm in the Amazon Web Services Management Console, the IAM CLI, or the IAM API, CloudWatch creates AWSServiceRoleForCloudWatchEvents and **AWSServiceRoleForCloudWatchAlarms_ActionSSM** for you.

The first time that you enable service and topology discovery, Application Signals creates **AWSServiceRoleForCloudWatchApplicationSignals** for you.

When you first enable an account to be a monitoring account for cross-account cross-Region functionality, CloudWatch creates **AWSServiceRoleForCloudWatchCrossAccount** for you.

When you first create an alarm that uses the `DB_PERF_INSIGHTS` metric math function, CloudWatch creates **AWSServiceRoleForCloudWatchMetrics_DbPerfInsights** for you.

For more information, see Creating a Service-Linked Role in the *IAM User Guide*.

Editing a service-linked role for CloudWatch

CloudWatch does not allow you to edit the **AWSServiceRoleForCloudWatchEvents**, **AWSServiceRoleForCloudWatchAlarms_ActionSSM**, **AWSServiceRoleForCloudWatchCrossAccount**, or **AWSServiceRoleForCloudWatchMetrics_DbPerfInsights** roles. After you create these roles, you cannot change their names because various entities might reference these roles. However, you can edit the description of these roles using IAM.

Editing a service-linked role description (IAM console)

You can use the IAM console to edit the description of a service-linked role.

To edit the description of a service-linked role (console)

1. In the navigation pane of the IAM console, choose **Roles**.
2. Choose the name of the role to modify.
3. To the far right of **Role description**, choose **Edit**.
4. Type a new description in the box, and choose **Save**.

Editing a service-linked role description (Amazon CLI)

You can use IAM commands from the Amazon Command Line Interface to edit the description of a service-linked role.

To change the description of a service-linked role (Amazon CLI)

1. (Optional) To view the current description for a role, use the following commands:

   ```
   $ aws iam get-role --role-name role-name
   ```

   Use the role name, not the ARN, to refer to roles with the Amazon CLI commands. For example, if a role has the following ARN: `arn:aws-cn:iam::123456789012:role/myrole`, you refer to the role as **`myrole`**.

2. To update a service-linked role's description, use the following command:

   ```
   $ aws iam update-role-description --role-name role-name --description description
   ```

Editing a service-linked role description (IAM API)

You can use the IAM API to edit the description of a service-linked role.

To change the description of a service-linked role (API)

1. (Optional) To view the current description for a role, use the following command:

   GetRole
2. To update a role's description, use the following command:

   UpdateRoleDescription

Deleting a service-linked role for CloudWatch

If you no longer have alarms that automatically stop, terminate, or reboot EC2 instances, we recommend that you delete the AWSServiceRoleForCloudWatchEvents role.

If you not longer have alarms that perform Systems Manager OpsCenter actions, we recommend that you delete the AWSServiceRoleForCloudWatchAlarms_ActionSSM role.

If you delete all alarms that use the `DB_PERF_INSIGHTS` metric math function, we recommend that you delete the **AWSServiceRoleForCloudWatchMetrics_DbPerfInsights** service-linked role.

That way you don’t have an unused entity that is not actively monitored or maintained. However, you must clean up your service-linked role before you can delete it.

Cleaning up a service-linked role

Before you can use IAM to delete a service-linked role, you must first confirm that the role has no active sessions and remove any resources used by the role.

To check whether the service-linked role has an active session in the IAM console

1. Open the IAM console at https://console.amazonaws.cn/iam/.
2. In the navigation pane, choose **Roles**. Choose the name (not the check box) of the AWSServiceRoleForCloudWatchEvents role.
3. On the **Summary** page for the selected role, choose **Access Advisor** and review the recent activity for the service-linked role.

> **Note**
>
> If you are unsure whether CloudWatch is using the AWSServiceRoleForCloudWatchEvents role, try to delete the role. If the service is using the role, then the deletion fails and you can view the Regions where the role is being used. If the role is being used, then you must wait for the session to end before you can delete the role. You cannot revoke the session for a service-linked role.

Deleting a service-linked role (IAM console)

You can use the IAM console to delete a service-linked role.

To delete a service-linked role (console)

1. Open the IAM console at https://console.amazonaws.cn/iam/.
2. In the navigation pane, choose **Roles**. Select the check box next to the name of the role you want to delete, not the name or row itself.
3. For **Role actions**, choose **Delete role**.
4. In the confirmation dialog box, review the service last accessed data, which shows when each of the selected roles last accessed an Amazon service. This helps you to confirm whether the role is currently active. To proceed, choose **Yes, Delete**.
5. Watch the IAM console notifications to monitor the progress of the service-linked role deletion. Because the IAM service-linked role deletion is asynchronous, the deletion task can succeed or fail after you submit the role for deletion. If the task fails, choose **View details** or **View Resources** from the notifications to learn why the deletion failed. If the deletion fails because there are resources in the service that are being used by the role, then the reason for the failure includes a list of resources.

Deleting a service-linked role (Amazon CLI)

You can use IAM commands from the Amazon Command Line Interface to delete a service-linked role.

To delete a service-linked role (Amazon CLI)

1. Because a service-linked role cannot be deleted if it is being used or has associated resources, you must submit a deletion request. That request can be denied if these conditions are not met. You must capture the `deletion-task-id` from the response to check the status of the deletion task. Type the following command to submit a service-linked role deletion request:

```
$ aws iam delete-service-linked-role --role-name service-linked-role-name
```

2. Type the following command to check the status of the deletion task:

```
$ aws iam get-service-linked-role-deletion-status --deletion-task-id deletion-task-id
```

   The status of the deletion task can be `NOT_STARTED`, `IN_PROGRESS`, `SUCCEEDED`, or `FAILED`. If the deletion fails, the call returns the reason that it failed so that you can troubleshoot.

Deleting a service-linked role (IAM API)

You can use the IAM API to delete a service-linked role.

To delete a service-linked role (API)

1. To submit a deletion request for a service-linked role, call [DeleteServiceLinkedRole](). In the request, specify the role name that you want to delete.

   Because a service-linked role cannot be deleted if it is being used or has associated resources, you must submit a deletion request. That request can be denied if these conditions are not met. You must capture the `DeletionTaskId` from the response to check the status of the deletion task.

2. To check the status of the deletion, call [GetServiceLinkedRoleDeletionStatus](). In the request, specify the `DeletionTaskId`.

   The status of the deletion task can be `NOT_STARTED`, `IN_PROGRESS`, `SUCCEEDED`, or `FAILED`. If the deletion fails, the call returns the reason that it failed so that you can troubleshoot.

CloudWatch updates to Amazon service-linked roles

View details about updates to Amazon managed policies for CloudWatch since this service began tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the CloudWatch Document history page.

Change	Description	Date
AWSServiceRoleForCloudWatchApplicationSignals – Update to permissions of service-linked role policy	Updated the CloudWatchApplicationSignalsServiceRolePolicy to exclude time windows from impacting the SLO attainment rate, error budget, and burn rate metrics. CloudWatch can retrieve exclusion windows on behalf of you.	March 13, 2025
AWSServiceRoleForObservabilityAdmin – New service-linked role	CloudWatch added this new service-linked role and corresponding managed policy, AWSObservabilityAdminServiceRolePolicy, to support resource and telemetry config discovery for Amazon Organizations.	November 26, 2024
AWSServiceRoleForCloudWatchApplicationSignals – Update to permissions of service-linked role policy	CloudWatch add more log groups to the scope of the `logs:StartQuery` and `logs:GetQueryResults` permissions granted by this role.	April 24, 2024
AWSServiceRoleForCloudWatchApplicationSignals – New service-linked role	CloudWatch added this new service-linked role to allow CloudWatch Application	November 9, 2023

Change	Description	Date
	Signals to collect CloudWatch Logs data, X-Ray trace data, CloudWatch metrics data, and tagging data from applications that you have enabled for CloudWatch Application Signals.	
AWSServiceRoleForCloudWatchMetrics_DbPerfInsights – New service-linked role	CloudWatch added this new service-linked role to allow CloudWatch to fetch Performance Insights metrics for alarming and snapshotting. An IAM policy is attached to this role, and the policy grants permission to CloudWatch to fetch Performance Insights metrics on your behalf.	September 13, 2023
AWSServiceRoleForCloudWatchAlarms_ActionSSMIncidents – New service-linked role	CloudWatch added a new service-linked role to allow CloudWatch to create incidents in Amazon Systems Manager Incident Manager.	April 26, 2021
CloudWatch started tracking changes	CloudWatch started tracking changes for its service-linked roles.	April 26, 2021

Using service-linked roles for CloudWatch RUM

CloudWatch RUM uses a Amazon Identity and Access Management (IAM) service-linked role. A service-linked role is a unique type of IAM role that is linked directly to RUM. The service-linked role is predefined by RUM and includes all the permissions that the service requires to call other Amazon services on your behalf.

RUM defines the permissions of the service-linked role, and unless defined otherwise, only RUM can assume the role. The defined permissions include the trust policy and the permissions policy, and that permissions policy cannot be attached to any other IAM entity.

You can delete the role only after first deleting its related resources. This restriction protects your RUM resources because you can't inadvertently remove permissions to access the resources.

For information about other services that support service-linked roles, see Amazon services that work with IAM and look for the services that have **Yes** in the **Service-linked roles** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

Service-linked role permissions for RUM

RUM uses the service-linked role named **AWSServiceRoleForCloudWatchRUM** – this role allows RUM to send Amazon X-Ray trace data into your account, for app monitors that you enable X-Ray tracing for.

The **AWSServiceRoleForCloudWatchRUM** service-linked role trusts the X-Ray service to assume the role. X-Ray sends the trace data to your account.

The **AWSServiceRoleForCloudWatchRUM** service-linked role has an IAM policy attached named **AmazonCloudWatchRUMServiceRolePolicy**. This policy grants permission to CloudWatch RUM to publish monitoring data to other relevant Amazon services. It includes permissions that allow RUM to complete the following actions:

- `xray:PutTraceSegments`
- `cloudwatch:PutMetricData`

The complete contents of **AmazonCloudWatchRUMServiceRolePolicy** are as follows.

```
{
 "Version": "2012-10-17",
```

```
  "Statement": [
   {
    "Effect": "Allow",
    "Action": [
     "xray:PutTraceSegments"
    ],
    "Resource": [
     "*"
    ]
   },
   {
    "Effect": "Allow",
    "Action": "cloudwatch:PutMetricData",
    "Resource": "*",
    "Condition": {
     "StringLike": {
      "cloudwatch:namespace": [
       "RUM/CustomMetrics/*",
       "AWS/RUM"
      ]
     }
    }
   }
  ]
 }
```

Creating a service-linked role for RUM

You do not need to manually create the service-linked role for CloudWatch RUM. The first time that you create an app monitor with X-Ray tracing enabled, or update an app monitor to use X-Ray tracing, RUM creates **AWSServiceRoleForCloudWatchRUM** for you.

For more information, see Creating a Service-Linked Role in the *IAM User Guide*.

Editing a service-linked role for RUM

CloudWatch RUM does not allow you to edit the **AWSServiceRoleForCloudWatchRUM** role. After you create these roles, you cannot change their names because various entities might reference these roles. However, you can edit the description of these roles using IAM.

Editing a service-linked role description (IAM console)

You can use the IAM console to edit the description of a service-linked role.

To edit the description of a service-linked role (console)

1. In the navigation pane of the IAM console, choose **Roles**.
2. Choose the name of the role to modify.
3. To the far right of **Role description**, choose **Edit**.
4. Type a new description in the box, and choose **Save**.

Editing a service-linked role description (Amazon CLI)

You can use IAM commands from the Amazon Command Line Interface to edit the description of a service-linked role.

To change the description of a service-linked role (Amazon CLI)

1. (Optional) To view the current description for a role, use the following commands:

   ```
   $ aws iam get-role --role-name role-name
   ```

   Use the role name, not the ARN, to refer to roles with the Amazon CLI commands. For example, if a role has the following ARN: `arn:aws-cn:iam::123456789012:role/myrole`, you refer to the role as **`myrole`**.
2. To update a service-linked role's description, use the following command:

   ```
   $ aws iam update-role-description --role-name role-name --description description
   ```

Editing a service-linked role description (IAM API)

You can use the IAM API to edit the description of a service-linked role.

To change the description of a service-linked role (API)

1. (Optional) To view the current description for a role, use the following command:

   GetRole
2. To update a role's description, use the following command:

   UpdateRoleDescription

Deleting a service-linked role for RUM

If you no longer have app monitors with X-Ray enabled, we recommend that you delete the **AWSServiceRoleForCloudWatchRUM** role.

That way you don't have an unused entity that is not actively monitored or maintained. However, you must clean up your service-linked role before you can delete it.

Cleaning up a service-linked role

Before you can use IAM to delete a service-linked role, you must first confirm that the role has no active sessions and remove any resources used by the role.

To check whether the service-linked role has an active session in the IAM console

1. Open the IAM console at https://console.amazonaws.cn/iam/.
2. In the navigation pane, choose **Roles**. Choose the name (not the check box) of the **AWSServiceRoleForCloudWatchRUM** role.
3. On the **Summary** page for the selected role, choose **Access Advisor** and review the recent activity for the service-linked role.

> **Note**
>
> If you are unsure whether RUM is using the **AWSServiceRoleForCloudWatchRUM** role, try to delete the role. If the service is using the role, then the deletion fails and you can view the Regions where the role is being used. If the role is being used, then you must wait for the session to end before you can delete the role. You cannot revoke the session for a service-linked role.

Deleting a service-linked role (IAM console)

You can use the IAM console to delete a service-linked role.

To delete a service-linked role (console)

1. Open the IAM console at https://console.amazonaws.cn/iam/.
2. In the navigation pane, choose **Roles**. Select the check box next to the name of the role you want to delete, not the name or row itself.

3. For **Role actions**, choose **Delete role**.
4. In the confirmation dialog box, review the service last accessed data, which shows when each of the selected roles last accessed an Amazon service. This helps you to confirm whether the role is currently active. To proceed, choose **Yes, Delete**.
5. Watch the IAM console notifications to monitor the progress of the service-linked role deletion. Because the IAM service-linked role deletion is asynchronous, the deletion task can succeed or fail after you submit the role for deletion. If the task fails, choose **View details** or **View Resources** from the notifications to learn why the deletion failed. If the deletion fails because there are resources in the service that are being used by the role, then the reason for the failure includes a list of resources.

Deleting a service-linked role (Amazon CLI)

You can use IAM commands from the Amazon Command Line Interface to delete a service-linked role.

To delete a service-linked role (Amazon CLI)

1. Because a service-linked role cannot be deleted if it is being used or has associated resources, you must submit a deletion request. That request can be denied if these conditions are not met. You must capture the `deletion-task-id` from the response to check the status of the deletion task. Type the following command to submit a service-linked role deletion request:

```
$ aws iam delete-service-linked-role --role-name service-linked-role-name
```

2. Type the following command to check the status of the deletion task:

```
$ aws iam get-service-linked-role-deletion-status --deletion-task-id deletion-task-id
```

The status of the deletion task can be `NOT_STARTED`, `IN_PROGRESS`, `SUCCEEDED`, or `FAILED`. If the deletion fails, the call returns the reason that it failed so that you can troubleshoot.

Deleting a service-linked role (IAM API)

You can use the IAM API to delete a service-linked role.

To delete a service-linked role (API)

1. To submit a deletion request for a service-linked role, call DeleteServiceLinkedRole. In the request, specify the role name that you want to delete.

   Because a service-linked role cannot be deleted if it is being used or has associated resources, you must submit a deletion request. That request can be denied if these conditions are not met. You must capture the `DeletionTaskId` from the response to check the status of the deletion task.

2. To check the status of the deletion, call GetServiceLinkedRoleDeletionStatus. In the request, specify the `DeletionTaskId`.

   The status of the deletion task can be `NOT_STARTED`, `IN_PROGRESS`, `SUCCEEDED`, or `FAILED`. If the deletion fails, the call returns the reason that it failed so that you can troubleshoot.

Supported Regions for CloudWatch RUM service-linked roles

CloudWatch RUM supports using service-linked roles in all of the Amazon Regions where the service is available. For more information, see CloudWatch RUM service endpoints.

Using service-linked roles for CloudWatch Application Insights

CloudWatch Application Insights uses Amazon Identity and Access Management (IAM) service-linked roles. A service-linked role is a unique type of IAM role that is linked directly to CloudWatch Application Insights. Service-linked roles are predefined by CloudWatch Application Insights and include all of the permissions that the service requires to call other Amazon services on your behalf.

A service-linked role makes setting up CloudWatch Application Insights easier because you don't have to manually add the necessary permissions. CloudWatch Application Insights defines the permissions of its service-linked roles, and unless defined otherwise, only CloudWatch Application Insights can assume its roles. The defined permissions include the trust policy and the permissions policy, and that permissions policy cannot be attached to any other IAM entity.

For information about other services that support service-linked roles, see Amazon Services That Work with IAM and look for the services that have **Yes** in the **Service-Linked Role** column. Choose a **Yes** link to view the service-linked role documentation for that service.

Service-linked role permissions for CloudWatch Application Insights

CloudWatch Application Insights uses the service-linked role named **AWSServiceRoleForApplicationInsights**. Application Insights uses this role to perform operations such as analyzing the resource groups of the customer, creating CloudFormation stacks to create alarms on metrics, and configuring the CloudWatch Agent on EC2 instances. This service-linked role has an IAM policy attached to it named `CloudwatchApplicationInsightsServiceLinkedRolePolicy`. For updates to this policy, see Application Insights updates to Amazon managed policies.

The role permissions policy allows CloudWatch Application Insights to complete the following actions on resources.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "CloudWatch",
      "Effect": "Allow",
      "Action": [
        "cloudwatch:DescribeAlarmHistory",
        "cloudwatch:DescribeAlarms",
        "cloudwatch:GetMetricData",
        "cloudwatch:ListMetrics",
        "cloudwatch:PutMetricAlarm",
        "cloudwatch:DeleteAlarms",
        "cloudwatch:PutAnomalyDetector",
        "cloudwatch:DeleteAnomalyDetector",
        "cloudwatch:DescribeAnomalyDetectors"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "CloudWatchLogs",
      "Effect": "Allow",
      "Action": [
        "logs:FilterLogEvents",
        "logs:GetLogEvents",
        "logs:DescribeLogStreams",
        "logs:DescribeLogGroups"
```

```
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "EventBridge",
      "Effect": "Allow",
      "Action": [
        "events:DescribeRule"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "CloudFormation",
      "Effect": "Allow",
      "Action": [
        "cloudFormation:CreateStack",
        "cloudFormation:UpdateStack",
        "cloudFormation:DeleteStack",
        "cloudFormation:DescribeStackResources",
        "cloudFormation:UpdateTerminationProtection"
      ],
      "Resource": [
        "arn:aws:cloudformation:*:*:stack/ApplicationInsights-*"
      ]
    },
    {
      "Sid": "CloudFormationStacks",
      "Effect": "Allow",
      "Action": [
        "cloudFormation:DescribeStacks",
        "cloudFormation:ListStackResources",
        "cloudFormation:ListStacks"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "Tag",
      "Effect": "Allow",
```

```
      "Action": [
        "tag:GetResources"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "ResourceGroups",
      "Effect": "Allow",
      "Action": [
        "resource-groups:ListGroupResources",
        "resource-groups:GetGroupQuery",
        "resource-groups:GetGroup"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "ApplicationInsightsResourceGroup",
      "Effect": "Allow",
      "Action": [
        "resource-groups:CreateGroup",
        "resource-groups:DeleteGroup"
      ],
      "Resource": [
        "arn:aws:resource-groups:*:*:group/ApplicationInsights-*"
      ]
    },
    {
      "Sid": "ElasticLoadBalancing",
      "Effect": "Allow",
      "Action": [
        "elasticloadbalancing:DescribeLoadBalancers",
        "elasticloadbalancing:DescribeTargetGroups",
        "elasticloadbalancing:DescribeTargetHealth"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "AutoScaling",
```

```
      "Effect": "Allow",
      "Action": [
        "autoscaling:DescribeAutoScalingGroups"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "SSMParameter",
      "Effect": "Allow",
      "Action": [
        "ssm:PutParameter",
        "ssm:DeleteParameter",
        "ssm:AddTagsToResource",
        "ssm:RemoveTagsFromResource",
        "ssm:GetParameters"
      ],
      "Resource": "arn:aws:ssm:*:*:parameter/AmazonCloudWatch-ApplicationInsights-*"
    },
    {
      "Sid": "SSMAssociation",
      "Effect": "Allow",
      "Action": [
        "ssm:CreateAssociation",
        "ssm:UpdateAssociation",
        "ssm:DeleteAssociation",
        "ssm:DescribeAssociation"
      ],
      "Resource": [
        "arn:aws:ec2:*:*:instance/*",
        "arn:aws:ssm:*:*:association/*",
        "arn:aws:ssm:*:*:managed-instance/*",
        "arn:aws:ssm:*:*:document/AWSEC2-
ApplicationInsightsCloudwatchAgentInstallAndConfigure",
        "arn:aws:ssm:*:*:document/AWS-ConfigureAWSPackage",
        "arn:aws:ssm:*:*:document/AmazonCloudWatch-ManageAgent"
      ]
    },
    {
      "Sid": "SSMOpsItem",
      "Effect": "Allow",
      "Action": [
        "ssm:GetOpsItem",
```

```
        "ssm:CreateOpsItem",
        "ssm:DescribeOpsItems",
        "ssm:UpdateOpsItem",
        "ssm:DescribeInstanceInformation"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "SSMTags",
      "Effect": "Allow",
      "Action": [
        "ssm:AddTagsToResource"
      ],
      "Resource": "arn:aws:ssm:*:*:opsitem/*"
    },
    {
      "Sid": "SSMGetCommandInvocation",
      "Effect": "Allow",
      "Action": [
        "ssm:ListCommandInvocations",
        "ssm:GetCommandInvocation"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "SSMSendCommand",
      "Effect": "Allow",
      "Action": "ssm:SendCommand",
      "Resource": [
        "arn:aws:ec2:*:*:instance/*",
        "arn:aws:ssm:*:*:document/AWSEC2-CheckPerformanceCounterSets",
        "arn:aws:ssm:*:*:document/AWS-ConfigureAWSPackage",
        "arn:aws:ssm:*:*:document/AWSEC2-DetectWorkload",
        "arn:aws:ssm:*:*:document/AmazonCloudWatch-ManageAgent"
      ]
    },
    {
      "Sid": "EC2",
      "Effect": "Allow",
      "Action": [
```

```
        "ec2:DescribeInstances",
        "ec2:DescribeVolumes",
        "ec2:DescribeVolumeStatus",
        "ec2:DescribeVpcs",
        "ec2:DescribeVpcAttribute",
        "ec2:DescribeNatGateways"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "RDS",
      "Effect": "Allow",
      "Action": [
        "rds:DescribeDBInstances",
        "rds:DescribeDBClusters"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "Lambda",
      "Effect": "Allow",
      "Action": [
        "lambda:ListFunctions",
        "lambda:GetFunctionConfiguration",
        "lambda:ListEventSourceMappings"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "EventBridgeManagedRule",
      "Effect": "Allow",
      "Action": [
        "events:PutRule",
        "events:PutTargets",
        "events:RemoveTargets",
        "events:DeleteRule"
      ],
      "Resource": [
```

```
        "arn:aws:events:*:*:rule/AmazonCloudWatch-ApplicationInsights-*"
      ]
    },
    {
      "Sid": "XRay",
      "Effect": "Allow",
      "Action": [
        "xray:GetServiceGraph",
        "xray:GetTraceSummaries",
        "xray:GetTimeSeriesServiceStatistics",
        "xray:GetTraceGraph"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "DynamoDB",
      "Effect": "Allow",
      "Action": [
        "dynamodb:ListTables",
        "dynamodb:DescribeTable",
        "dynamodb:DescribeContributorInsights",
        "dynamodb:DescribeTimeToLive"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "ApplicationAutoscaling",
      "Effect": "Allow",
      "Action": [
        "application-autoscaling:DescribeScalableTargets"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "S3",
      "Effect": "Allow",
      "Action": [
        "s3:ListAllMyBuckets",
```

```
        "s3:GetMetricsConfiguration",
        "s3:GetReplicationConfiguration"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "States",
      "Effect": "Allow",
      "Action": [
        "states:ListStateMachines",
        "states:DescribeExecution",
        "states:DescribeStateMachine",
        "states:GetExecutionHistory"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "APIGateway",
      "Effect": "Allow",
      "Action": [
        "apigateway:GET"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "ECS",
      "Effect": "Allow",
      "Action": [
        "ecs:DescribeClusters",
        "ecs:DescribeContainerInstances",
        "ecs:DescribeServices",
        "ecs:DescribeTaskDefinition",
        "ecs:DescribeTasks",
        "ecs:DescribeTaskSets",
        "ecs:ListClusters",
        "ecs:ListContainerInstances",
        "ecs:ListServices",
        "ecs:ListTasks"
```

```
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "ECSCluster",
      "Effect": "Allow",
      "Action": [
        "ecs:UpdateClusterSettings"
      ],
      "Resource": [
        "arn:aws:ecs:*:*:cluster/*"
      ]
    },
    {
      "Sid": "EKS",
      "Effect": "Allow",
      "Action": [
        "eks:DescribeCluster",
        "eks:DescribeFargateProfile",
        "eks:DescribeNodegroup",
        "eks:ListClusters",
        "eks:ListFargateProfiles",
        "eks:ListNodegroups",
        "fsx:DescribeFileSystems",
        "fsx:DescribeVolumes"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "SNS",
      "Effect": "Allow",
      "Action": [
        "sns:GetSubscriptionAttributes",
        "sns:GetTopicAttributes",
        "sns:GetSMSAttributes",
        "sns:ListSubscriptionsByTopic",
        "sns:ListTopics"
      ],
      "Resource": [
        "*"
```

```
      ]
    },
    {
      "Sid": "SQS",
      "Effect": "Allow",
      "Action": [
        "sqs:ListQueues"
      ],
      "Resource": "*"
    },
    {
      "Sid": "CloudWatchLogsDeleteSubscriptionFilter",
      "Effect": "Allow",
      "Action": [
        "logs:DeleteSubscriptionFilter"
      ],
      "Resource": [
        "arn:aws:logs:*:*:log-group:*"
      ]
    },
    {
      "Sid": "CloudWatchLogsCreateSubscriptionFilter",
      "Effect": "Allow",
      "Action": [
        "logs:PutSubscriptionFilter"
      ],
      "Resource": [
        "arn:aws:logs:*:*:log-group:*",
        "arn:aws:logs:*:*:destination:AmazonCloudWatch-ApplicationInsights-
LogIngestionDestination*"
      ]
    },
    {
      "Sid": "EFS",
      "Effect": "Allow",
      "Action": [
        "elasticfilesystem:DescribeFileSystems"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "Route53",
```

```
      "Effect": "Allow",
      "Action": [
        "route53:GetHostedZone",
        "route53:GetHealthCheck",
        "route53:ListHostedZones",
        "route53:ListHealthChecks",
        "route53:ListQueryLoggingConfigs"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "Route53Resolver",
      "Effect": "Allow",
      "Action": [
        "route53resolver:ListFirewallRuleGroupAssociations",
        "route53resolver:GetFirewallRuleGroup",
        "route53resolver:ListFirewallRuleGroups",
        "route53resolver:ListResolverEndpoints",
        "route53resolver:GetResolverQueryLogConfig",
        "route53resolver:ListResolverQueryLogConfigs",
        "route53resolver:ListResolverQueryLogConfigAssociations",
        "route53resolver:GetResolverEndpoint",
        "route53resolver:GetFirewallRuleGroupAssociation"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

You must configure permissions to allow an IAM entity (such as a user, group, or role) to create, edit, or delete a service-linked role. For more information, see Service-Linked Role Permissions in the *IAM User Guide*.

Creating a service-linked role for CloudWatch Application Insights

You don't need to manually create a service-linked role. When you create a new Application Insights application in the Amazon Web Services Management Console, CloudWatch Application Insights creates the service-linked role for you.

If you delete this service-linked role, and then want to create it again, you can use the same process to recreate the role in your account. When you create a new Application Insights application, CloudWatch Application Insights creates the service-linked role for you again.

Editing a service-linked role for CloudWatch Application Insights

CloudWatch Application Insights does not allow you to edit the AWSServiceRoleForApplicationInsights service-linked role. After you create a service-linked role, you cannot change the name of the role because various entities might reference the role. However, you can edit the description of the role using IAM. For more information, see Editing a Service-Linked Role in the *IAM User Guide*.

Deleting a service-linked role for CloudWatch Application Insights

If you no longer need to use a feature or service that requires a service-linked role, we recommend that you delete that role. That way you avoid having an unused entity that is not actively monitored or maintained. However, you must delete all applications in Application Insights before you can manually delete the role.

> **Note**
>
> If the CloudWatch Application Insights service is using the role when you try to delete the resources, the deletion might fail. If that happens, wait for a few minutes and try the operation again.

To delete CloudWatch Application Insights resources used by the AWSServiceRoleForApplicationInsights

- Delete all of your CloudWatch Application Insights applications. For more information, see "Deleting Your Application(s)" in the CloudWatch Application Insights User Guide.

To manually delete the service-linked role using IAM

Use the IAM console, the Amazon CLI, or the Amazon API to delete the AWSServiceRoleForApplicationInsights service-linked role. For more information, see Deleting a Service-Linked Role in the *IAM User Guide*.

Supported Regions for CloudWatch Application Insights service-linked roles

CloudWatch Application Insights supports using service-linked roles in all of the Amazon Regions where the service is available. For more information, see CloudWatch Application Insights Regions and Endpoints.

Amazon managed policies for Amazon CloudWatch Application Insights

An Amazon managed policy is a standalone policy that is created and administered by Amazon. Amazon managed policies are designed to provide permissions for many common use cases so that you can start assigning permissions to users, groups, and roles.

Keep in mind that Amazon managed policies might not grant least-privilege permissions for your specific use cases because they're available for all Amazon customers to use. We recommend that you reduce permissions further by defining customer managed policies that are specific to your use cases.

You cannot change the permissions defined in Amazon managed policies. If Amazon updates the permissions defined in an Amazon managed policy, the update affects all principal identities (users, groups, and roles) that the policy is attached to. Amazon is most likely to update an Amazon managed policy when a new Amazon Web Services service is launched or new API operations become available for existing services.

For more information, see Amazon managed policies in the *IAM User Guide*.

Amazon managed policy: CloudWatchApplicationInsightsFullAccess

You can attach the `CloudWatchApplicationInsightsFullAccess` policy to your IAM identities.

This policy grants administrative permissions that allow full access to Application Insights functionality.

Permissions details

This policy includes the following permissions.

- `applicationinsights` – Allows full access to Application Insights functionality.
- `iam` – Allows Application Insights to create the service-linked role, AWSServiceRoleForApplicationInsights. This is required so that Application Insights can perform operations such as analyze the resource groups of a customer, create CloudFormation stacks to create alarms on metrics, and configure the CloudWatch Agent on EC2 instances. For more information, see Using service-linked roles for CloudWatch Application Insights.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "applicationinsights:*",
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "ec2:DescribeInstances",
        "ec2:DescribeVolumes",
        "rds:DescribeDBInstances",
        "rds:DescribeDBClusters",
        "sqs:ListQueues",
        "elasticloadbalancing:DescribeLoadBalancers",
        "elasticloadbalancing:DescribeTargetGroups",
        "elasticloadbalancing:DescribeTargetHealth",
        "autoscaling:DescribeAutoScalingGroups",
        "lambda:ListFunctions",
        "dynamodb:ListTables",
        "s3:ListAllMyBuckets",
        "sns:ListTopics",
```

```
        "states:ListStateMachines",
        "apigateway:GET",
        "ecs:ListClusters",
        "ecs:DescribeTaskDefinition",
        "ecs:ListServices",
        "ecs:ListTasks",
        "eks:ListClusters",
        "eks:ListNodegroups",
        "fsx:DescribeFileSystems",
        "logs:DescribeLogGroups",
        "elasticfilesystem:DescribeFileSystems"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "iam:CreateServiceLinkedRole"
      ],
      "Resource": [
        "arn:aws:iam::*:role/aws-service-role/application-insights.amazonaws.com/
AWSServiceRoleForApplicationInsights"
      ],
      "Condition": {
        "StringEquals": {
          "iam:AWSServiceName": "application-insights.amazonaws.com"
        }
      }
    }
  ]
}
```

Amazon managed policy: CloudWatchApplicationInsightsReadOnlyAccess

You can attach the `CloudWatchApplicationInsightsReadOnlyAccess` policy to your IAM identities.

This policy grants administrative permissions that allow read-only access to all Application Insights functionality.

Permissions details

This policy includes the following permissions.

- `applicationinsights` – Allows read-only access to Application Insights functionality.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "applicationinsights:Describe*",
                "applicationinsights:List*"
            ],
            "Resource": "*"
        }
    ]
}
```

Amazon managed policy: CloudwatchApplicationInsightsServiceLinkedRolePolicy

You can't attach CloudwatchApplicationInsightsServiceLinkedRolePolicy to your IAM entities. This policy is attached to a service-linked role that allows Application Insights to monitor customer resources. For more information, see Using service-linked roles for CloudWatch Application Insights.

Application Insights updates to Amazon managed policies

View details about updates to Amazon managed policies for Application Insights since this service began tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the Application Insights Document history page.

Change	Description	Date
CloudwatchApplicationInsightsServiceLinkedRolePolicy – Update to an existing policy	Application Insights added a new permission. The policy change allows Amazon CloudWatch Application Insights to enable and disable termination protection on CloudFormation stacks to manage SSM resources used to install and configure CloudWatch agents.	July 25, 2024
CloudwatchApplicationInsightsServiceLinkedRolePolicy – Update to an existing policy	Application Insights added new permissions to list CloudFormation stacks. These permissions are required for Amazon CloudWatch Application Insights to analyze and monitor Amazon resources nested in the CloudFormation stack.	April 24, 2023
CloudwatchApplicationInsightsServiceLinkedRolePolicy – Update to an existing policy	Application Insights added new permissions to get list of Amazon VPC and Route 53 resources. These permissions are required for Amazon CloudWatch Application Insights to automatically set up best practice network	January 23, 2023

Change	Description	Date
	monitoring with Amazon CloudWatch.	
CloudwatchApplicationInsightsServiceLinkedRolePolicy – Update to an existing policy	Application Insights added new permissions to get SSM command invocation results. These permissions are required for Amazon CloudWatch Application Insights to automatically detect and monitor workloads running on Amazon EC2 instances.	December 19, 2022
CloudwatchApplicationInsightsServiceLinkedRolePolicy – Update to an existing policy	Application Insights added new permissions to describe Amazon VPC and Route 53 resources. These permissions are required for Amazon CloudWatch Application Insights to read customer Amazon VPC and Route 53 resource configurations, and to help customers automatically set up best practice network monitoring with Amazon CloudWatch.	December 19, 2022

Change	Description	Date
CloudwatchApplicationInsightsServiceLinkedRolePolicy – Update to an existing policy	Application Insights added new permissions to describe EFS resources. These permissions are required for Amazon CloudWatch Application Insights to read Amazon EFS customer resource configurations, and to help customers automatically set up best practices for EFS monitoring with CloudWatch.	October 3, 2022
CloudwatchApplicationInsightsServiceLinkedRolePolicy – Update to an existing policy	Application Insights added new permissions to describe the EFS file system. These permissions are required for Amazon CloudWatch Application Insights to create account-based applications by querying all of the supported resources in an account.	October 3, 2022

Change	Description	Date
CloudwatchApplicationInsightsServiceLinkedRolePolicy – Update to an existing policy	Application Insights added new permissions to retrieve information about FSx resources. These permissions are required for Amazon CloudWatch Application Insights to monitor workloads by retrieving sufficient information about the underlying FSx volumes.	September 12, 2022
Amazon managed policy: CloudWatchApplicationInsightsFullAccess – Update to an existing policy	Application Insights added a new permission to describe log groups. This permissions is required for Amazon CloudWatch Application Insights to ensure that the correct permissions for monitoring log groups are in an account when creating a new application.	January 24, 2022

Change	Description	Date
CloudwatchApplicationInsightsServiceLinkedRolePolicy – Update to an existing policy	Application Insights added new permissions to create and delete CloudWatch Log Subscription Filters. These permissions are required for Amazon CloudWatch Application Insights to create Subscription Filters to facilitate log monitoring of resources within configured applications.	January 24, 2022
CloudwatchApplicationInsightsServiceLinkedRolePolicy – Update to an existing policy	Application Insights added new permissions to describe target groups and target health for Elastic Load Balancers. These permissions are required for Amazon CloudWatch Application Insights to create account-based applications by querying all of the supported resources in an account.	November 4, 2021

Change	Description	Date
CloudwatchApplicationInsightsServiceLinkedRolePolicy – Update to an existing policy	Application Insights added new permissions to run the `AmazonCloudWatch-ManageAgent` SSM document on Amazon EC2 instances. This permissions is required for Amazon CloudWatch Application Insights to clean up CloudWatch agent configuration files created by Application Insights.	September 30, 2021

Change	Description	Date
CloudwatchApplicationInsightsServiceLinkedRolePolicy – Update to an existing policy	Application Insights added new permissions to support account-based application monitoring to onboard and monitor all supported resources in your account. These permissions are required for Amazon CloudWatch Application Insights to query, tag resources, and create groups for these resources. Application Insights added new permissions to support monitoring of SNS topics. These permissions are required for Amazon CloudWatch Application Insights to gather metadata from SNS resources to configure monitoring for SNS topics.	September 15, 2021

Change	Description	Date
Amazon managed policy: CloudWatchApplicationInsightsFullAccess – Update to an existing policy	Application Insights added new permissions to describe and list supported resources. These permissions are required for Amazon CloudWatch Application Insights to create account-based applications by querying all of the supported resources in an account.	September 15, 2021
CloudwatchApplicationInsightsServiceLinkedRolePolicy – Update to an existing policy	Application Insights added new permissions to describe FSx resources. These permissions are required for Amazon CloudWatch Application Insights to read customer FSx resource configurations, and to help customers automatically set up best practice FSx monitoring with CloudWatch.	August 31, 2021

Change	Description	Date
CloudwatchApplicationInsightsServiceLinkedRolePolicy – Update to an existing policy	Application Insights added new permissions to describe and list ECS and EKS service resources. This permission is required for Amazon CloudWatch Application Insights to read customer container resources configuration, and to help customers automatically set up best practice container monitoring with CloudWatch.	May 18, 2021
CloudwatchApplicationInsightsServiceLinkedRolePolicy – Update to an existing policy	Application Insights added new permissions to allow OpsCenter to tag OpsItems using the `ssm:AddTagsToResource` action on resources with the `opsitem` resource type. This permission is required by OpsCenter. Amazon CloudWatch Application Insights creates OpsItems so that the customer can resolve problems using Amazon SSM OpsCenter.	April 13, 2021
Application Insights started tracking changes	Application Insights started tracking changes for its Amazon managed policies.	April 13, 2021

Amazon CloudWatch permissions reference

The following table lists each CloudWatch API operation and the corresponding actions for which you can grant permissions to perform the action. You specify the actions in the policy's `Action` field, and you specify a wildcard character (*) as the resource value in the policy's `Resource` field.

You can use Amazon-wide condition keys in your CloudWatch policies to express conditions. For a complete list of Amazon-wide keys, see Amazon Global and IAM Condition Context Keys in the *IAM User Guide*.

> **Note**
>
> To specify an action, use the `cloudwatch:` prefix followed by the API operation name. For example: `cloudwatch:GetMetricData`, `cloudwatch:ListMetrics`, or `cloudwatch:*` (for all CloudWatch actions).

Topics

CloudWatch API operations and required permissions for actions

CloudWatch API operations	Required permissions (API actions)
DeleteAlarms	`cloudwatch:DeleteAlarms` Required to delete an alarm.
DeleteDashboards	`cloudwatch:DeleteDashboards`

CloudWatch API operations	Required permissions (API actions)
	Required to delete a dashboard.
[DeleteMetricStream](https://docs.aws.amazon.com/AmazonCloudWatch/latest/APIReference/API_DeleteMetricStream.html)	`cloudwatch:DeleteMetricStream` Required to delete a metric stream.
DescribeAlarmHistory	`cloudwatch:DescribeAlarmHistory` Required to view alarm history. To retrieve information about composite alarms, your `cloudwatch:DescribeAlarmHistory` permission must have a * scope. You can't return information about composite alarms if your `cloudwatch:DescribeAlarmHistory` permission has a narrower scope.
DescribeAlarms	`cloudwatch:DescribeAlarms` Required to retrieve information about alarms. To retrieve information about composite alarms, your `cloudwatch:DescribeAlarms` permission must have a * scope. You can't return information about composite alarms if your `cloudwatch:DescribeAlarms` permission has a narrower scope.
DescribeAlarmsForMetric	`cloudwatch:DescribeAlarmsForMetric` Required to view alarms for a metric.

CloudWatch API operations	Required permissions (API actions)
DisableAlarmActions	`cloudwatch:DisableAlarmActions` Required to disable an alarm action.
EnableAlarmActions	`cloudwatch:EnableAlarmActions` Required to enable an alarm action.
GetDashboard	`cloudwatch:GetDashboard` Required to display data about existing dashboards.
GetMetricData	`cloudwatch:GetMetricData` Required to graph metric data in the CloudWatch console, to retrieve large batches of metric data, and perform metric math on that data.
GetMetricStatistics	`cloudwatch:GetMetricStatistics` Required to view graphs in other parts of the CloudWatch console and in dashboard widgets.
GetMetricStream	`cloudwatch:GetMetricStream` Required to view information about a metric stream.

CloudWatch API operations	Required permissions (API actions)
[GetMetricWidgetImage](#)	`cloudwatch:GetMetricWidgetImage` Required to retrieve a snapshot graph of one or more CloudWatch metrics as a bitmap image.
[ListDashboards](#)	`cloudwatch:ListDashboards` Required to view the list of CloudWatch dashboards in your account.
ListEntitiesForMetric (CloudWatch console-only permission)	`cloudwatch:ListEntitiesForMetric` Required to find the entities associated with a metric. Required to explore related telemetry within the CloudWatch console.
[ListMetrics](#)	`cloudwatch:ListMetrics` Required to view or search metric names within the CloudWatch console and in the CLI. Required to select metrics on dashboard widgets.
[ListMetricStreams](#)	`cloudwatch:ListMetricStreams` Required to view or search the list of metric streams in the account.

CloudWatch API operations	Required permissions (API actions)
PutCompositeAlarm	`cloudwatch:PutCompositeAlarm` Required to create a composite alarm. To create a composite alarm, your `cloudwatch:PutCompositeAlarm` permission must have a * scope. You can't return information about composite alarms if your `cloudwatch:PutCompositeAlarm` permission has a narrower scope.
PutDashboard	`cloudwatch:PutDashboard` Required to create a dashboard or update an existing dashboard.
PutMetricAlarm	`cloudwatch:PutMetricAlarm` Required to create or update an alarm.
PutMetricData	`cloudwatch:PutMetricData` Required to create metrics.
PutMetricStream	`cloudwatch:PutMetricStream` Required to create a metric stream.
SetAlarmState	`cloudwatch:SetAlarmState` Required to manually set an alarm's state.

CloudWatch API operations	Required permissions (API actions)
StartMetricStreams	`cloudwatch:StartMetricStreams` Required to start the flow of metrics in a metric stream.
StopMetricStreams	`cloudwatch:StopMetricStreams` Required to temporarily stop the flow of metrics in a metric stream.
TagResource	`cloudwatch:TagResource` Required to add or update tags on CloudWatch resources such as alarms and Contributor Insights rules.
UntagResource	`cloudwatch:UntagResource` Required to remove tags from CloudWatch resources .

CloudWatch Application Signals API operations and required permissions for actions

CloudWatch Application Signals API operations	Required permissions (API actions)
BatchGetServiceLevelObjectiveBudgetReport	`application-signals:BatchGetServiceLevelObjectiveBudgetReport`

CloudWatch Application Signals API operations	Required permissions (API actions)
	Required to retrieve service level objective budget reports.
[CreateServiceLevelObjective](#)	`application-signals:CreateServiceLevelObjective` Required to create a service level objective (SLO).
[DeleteServiceLevelObjective](#)	`application-signals:DeleteServiceLevelObjective` Required to delete a service level objective (SLO).
[GetService](#)	`application-signals:GetService` Required to retrieve information about a service discovered by Application Signals.
[GetServiceLevelObjective](#)	`application-signals:GetServiceLevelObjective` Required to retrieve information about a service level objective (SLO).
ListObservedEntities	`application-signals:ListObservedEntities` Grants permission to list entities that are associated with other entities.

CloudWatch Application Signals API operations	Required permissions (API actions)
ListServiceDependencies	`application-signals:ListServiceDependencies` Required to retrieve a list of service dependencies of a service that you specify. This service and the dependencies were discovered by Application Signals.
ListServiceDependents	`application-signals:ListServiceDependents` Required to retrieve a list of dependents that invoked a service that you specify. This service and the dependents were discovered by Application Signals.
ListServiceLevelObjectives	`application-signals:ListServiceLevelObjectives` Required to retrieve a list of service level objectives (SLOs) in the account.
ListServiceOperations	`application-signals:ListServiceOperations` Required to retrieve a list of service operations of a service that you specify. This service and the dependencies were discovered by Application Signals.

CloudWatch Application Signals API operations	Required permissions (API actions)
ListServices	`application-signals:ListServices` Required to retrieve a list of services discovered by Application Signals.
ListTagsForResource	`application-signals:ListTagsForResource` Required to retrieve a list of the tags associated with a resource.
StartDiscovery	`application-signals:StartDiscovery` Required to be able to enable Application Signals in the account and create the required service-linked role.
TagResource	`application-signals:TagResource` Required to be able to add tags to resources.
UntagResource	`application-signals:UntagResource` Required to be able to remove tags from resources.

CloudWatch Application Signals API operations	Required permissions (API actions)
UpdateServiceLevelObjective	`application-signals:UpdateServiceLevelObjective` Required to update an existing service level objective

CloudWatch Contributor Insights API operations and required permissions for actions

Important

When you grant a user the `cloudwatch:PutInsightRule` permission, by default that user can create a rule that evaluates any log group in CloudWatch Logs. You can add IAM policy conditions that limit these permissions for a user to include and exclude specific log groups. For more information, see Using condition keys to limit Contributor Insights users' access to log groups.

CloudWatch Contributor Insights API operations	Required permissions (API actions)
DeleteInsightRules	`cloudwatch:DeleteInsightRules` Required to delete Contributor Insights rules.
DescribeInsightRules	`cloudwatch:DescribeInsightRules` Required to view the Contributor Insights rules in your account.
EnableInsightRules	`cloudwatch:EnableInsightRules`

CloudWatch Contributor Insights API operations	Required permissions (API actions)
	Required to enable Contributor Insights rules.
[GetInsightRuleReport]()	`cloudwatch:GetInsightRuleReport` Required to retrieve time series data and other statistics collectd by Contributor Insights rules.
[PutInsightRule]()	`cloudwatch:PutInsightRule` Required to create Contributor Insights rules. See the **Important** note at the beginning of this table.

CloudWatch Events API operations and required permissions for actions

CloudWatch Events API operations	Required permissions (API actions)
[DeleteRule]()	`events:DeleteRule` Required to delete a rule.
[DescribeRule]()	`events:DescribeRule` Required to list the details about a rule.
[DisableRule]()	`events:DisableRule` Required to disable a rule.
[EnableRule]()	`events:EnableRule`

CloudWatch Events API operations	Required permissions (API actions)
	Required to enable a rule.
ListRuleNamesByTarget	`events:ListRuleNamesByTarget` Required to list rules associated with a target.
ListRules	`events:ListRules` Required to list all rules in your account.
ListTargetsByRule	`events:ListTargetsByRule` Required to list all targets associated with a rule.
PutEvents	`events:PutEvents` Required to add custom events that can be matched to rules.
PutRule	`events:PutRule` Required to create or update a rule.
PutTargets	`events:PutTargets` Required to add targets to a rule.
RemoveTargets	`events:RemoveTargets` Required to remove a target from a rule.

CloudWatch Events API operations	Required permissions (API actions)
TestEventPattern	`events:TestEventPattern` Required to test an event pattern against a given event.

CloudWatch Logs API operations and required permissions for actions

> **Note**
>
> CloudWatch Logs permissions can be found in the CloudWatch Logs user guide.

Amazon EC2 API operations and required permissions for actions

Amazon EC2 API operations	Required permissions (API actions)
DescribeInstanceStatus	`ec2:DescribeInstanceStatus` Required to view EC2 instance status details.
DescribeInstances	`ec2:DescribeInstances` Required to view EC2 instance details.
RebootInstances	`ec2:RebootInstances` Required to reboot an EC2 instance.
StopInstances	`ec2:StopInstances` Required to stop an EC2 instance.
TerminateInstances	

Amazon EC2 API operations	Required permissions (API actions)
	`ec2:TerminateInstances` Required to terminate an EC2 instance.

Amazon EC2 Auto Scaling API operations and required permissions for actions

Amazon EC2 Auto Scaling API operations	Required permissions (API actions)
Scaling	`autoscaling:Scaling` Required to scale an Auto Scaling group.
Trigger	`autoscaling:Trigger` Required to trigger an Auto Scaling action.

Compliance validation for Amazon CloudWatch

To learn whether an Amazon Web Services service is within the scope of specific compliance programs, see Amazon Web Services services in Scope by Compliance Program and choose the compliance program that you are interested in. For general information, see Amazon Web Services Compliance Programs.

You can download third-party audit reports using Amazon Artifact. For more information, see Downloading Reports in Amazon Artifact.

Your compliance responsibility when using Amazon Web Services services is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. Amazon provides the following resources to help with compliance:

- Security & Compliance – These solution implementation guides discuss architectural considerations and provide steps for deploying security and compliance features.
- Amazon Compliance Resources – This collection of workbooks and guides might apply to your industry and location.

- Evaluating Resources with Rules in the *Amazon Config Developer Guide* – The Amazon Config service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- Amazon Security Hub – This Amazon Web Services service provides a comprehensive view of your security state within Amazon. Security Hub uses security controls to evaluate your Amazon resources and to check your compliance against security industry standards and best practices. For a list of supported services and controls, see Security Hub controls reference.
- Amazon GuardDuty – This Amazon Web Services service detects potential threats to your Amazon Web Services accounts, workloads, containers, and data by monitoring your environment for suspicious and malicious activities. GuardDuty can help you address various compliance requirements, like PCI DSS, by meeting intrusion detection requirements mandated by certain compliance frameworks.

Resilience in Amazon CloudWatch

The Amazon global infrastructure is built around Amazon Regions and Availability Zones. Regions provide multiple physically separated and isolated Availability Zones, which are connected through low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

For more information about Amazon Regions and Availability Zones, see Amazon Global Infrastructure.

Infrastructure security in Amazon CloudWatch

As a managed service, Amazon CloudWatch is protected by Amazon global network security. For information about Amazon security services and how Amazon protects infrastructure, see Amazon Cloud Security. To design your Amazon environment using the best practices for infrastructure security, see Infrastructure Protection in *Security Pillar Amazon Well‐Architected Framework*.

You use Amazon published API calls to access CloudWatch through the network. Clients must support the following:

- Transport Layer Security (TLS). We require TLS 1.2 and recommend TLS 1.3.

- Cipher suites with perfect forward secrecy (PFS) such as DHE (Ephemeral Diffie-Hellman) or ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the Amazon Security Token Service (Amazon STS) to generate temporary security credentials to sign requests.

Network isolation

A virtual private cloud (VPC) is a virtual network in your own logically isolated area in the Amazon Web Services Cloud. A subnet is a range of IP addresses in a VPC. You can deploy a variety of Amazon resources in the subnets of your VPCs. For example, you can deploy Amazon EC2 instances, EMR clusters, and DynamoDB tables in subnets. For more information, see the Amazon VPC User Guide.

To enable CloudWatch to communicate with resources in a VPC without going through the public internet, use Amazon PrivateLink. For more information, see Using CloudWatch, CloudWatch Synthetics, and CloudWatch Network Monitoring with interface VPC endpoints.

A private subnet is a subnet with no default route to the public internet. Deploying an Amazon resource in a private subnet does not prevent Amazon CloudWatch from collecting built-in metrics from the resource.

If you need to publish custom metrics from an Amazon resource in a private subnet, you can do so using a proxy server. The proxy server forwards those HTTPS requests to the public API endpoints for CloudWatch.

Amazon Security Hub

Monitor your usage of CloudWatch as it relates to security best practices by using Amazon Security Hub. Security Hub uses *security controls* to evaluate resource configurations and *security standards* to help you comply with various compliance frameworks. For more information about using Security Hub to evaluate CloudWatch resources, see Amazon CloudWatch controls in the Amazon Security Hub User Guide.

Using CloudWatch, CloudWatch Synthetics, and CloudWatch Network Monitoring with interface VPC endpoints

If you use Amazon Virtual Private Cloud (Amazon VPC) to host your Amazon resources, you can establish a private connection between your VPC, CloudWatch, CloudWatch Synthetics, and CloudWatch Network Monitoring features. You can use these connections to enable these services to communicate with resources in your VPC without going through the public internet.

Amazon VPC is an Amazon service that you can use to launch Amazon resources in a virtual network that you define. With a VPC, you have control over your network settings, such the IP address range, subnets, route tables, and network gateways. To connect your VPC to CloudWatch services, you define an *interface VPC endpoint* for your VPC. The endpoint provides reliable, scalable connectivity to CloudWatch and supported CloudWatch services without requiring an internet gateway, network address translation (NAT) instance, or VPN connection. For more information, see What is Amazon VPC in the *Amazon VPC User Guide*.

Interface VPC endpoints are powered by Amazon PrivateLink, an Amazon technology that enables private communication between Amazon services using an elastic network interface with private IP addresses. For more information, see the following blog post: New – Amazon PrivateLink for Amazon Services

The following steps are for users of Amazon VPC. For more information, see Getting Started in the *Amazon VPC User Guide*.

CloudWatch VPC endpoints

CloudWatch currently supports VPC endpoints in the following Amazon Regions:

- US East (Ohio)
- US East (N. Virginia)
- US West (N. California)
- US West (Oregon)
- Asia Pacific (Hong Kong)
- Asia Pacific (Mumbai)
- Asia Pacific (Osaka)
- Asia Pacific (Seoul)

- Asia Pacific (Singapore)
- Asia Pacific (Sydney)
- Asia Pacific (Tokyo)
- Canada (Central)
- Europe (Frankfurt)
- Europe (Ireland)
- Europe (London)
- Europe (Paris)
- Middle East (UAE)
- South America (São Paulo)
- Amazon GovCloud (US-East)
- Amazon GovCloud (US-West)

Creating a VPC endpoint for CloudWatch

To start using CloudWatch with your VPC, create an interface VPC endpoint for CloudWatch. The service name to choose is `com.amazonaws.region.monitoring`. For more information, see Creating an interface endpoint in the *Amazon VPC User Guide*.

You do not need to change the settings for CloudWatch. CloudWatch calls other Amazon services using either public endpoints or private interface VPC endpoints, whichever are in use. For example, if you create an interface VPC endpoint for CloudWatch, and you already have metrics flowing to CloudWatch from resources located on your VPC, these metrics begin flowing through the interface VPC endpoint by default.

Controlling access to your CloudWatch VPC endpoint

A VPC endpoint policy is an IAM resource policy that you attach to an endpoint when you create or modify the endpoint. If you don't attach a policy when you create an endpoint, Amazon VPC attaches a default policy for you that allows full access to the service. An endpoint policy doesn't override or replace user policies or service-specific policies. It's a separate policy for controlling access from the endpoint to the specified service.

Endpoint policies must be written in JSON format.

For more information, see Controlling access to services with VPC endpoints in the *Amazon VPC User Guide*.

The following is an example of an endpoint policy for CloudWatch. This policy allows users connecting to CloudWatch through the VPC to send metric data to CloudWatch and prevents them from performing other CloudWatch actions.

```
{
  "Statement": [
    {
      "Sid": "PutOnly",
      "Principal": "*",
      "Action": [
        "cloudwatch:PutMetricData"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

To edit the VPC endpoint policy for CloudWatch

1. Open the Amazon VPC console at https://console.amazonaws.cn/vpc/.
2. In the navigation pane, choose **Endpoints**.
3. If you have not already created the endpoint for CloudWatch, choose **Create endpoint**. Select **com.amazonaws.*region*.monitoring**, and then choose **Create endpoint**.
4. Select the **com.amazonaws.*region*.monitoring** endpoint, and then choose the **Policy** tab.
5. Choose **Edit policy**, and then make your changes.

CloudWatch Synthetics VPC endpoint

CloudWatch Synthetics currently supports VPC endpoints in the following Amazon Regions:

- US East (Ohio)
- US East (N. Virginia)
- US West (N. California)
- US West (Oregon)

- Asia Pacific (Hong Kong)
- Asia Pacific (Mumbai)
- Asia Pacific (Seoul)
- Asia Pacific (Singapore)
- Asia Pacific (Sydney)
- Asia Pacific (Tokyo)
- Canada (Central)
- Europe (Frankfurt)
- Europe (Ireland)
- Europe (London)
- Europe (Paris)
- South America (São Paulo)

Creating a VPC endpoint for CloudWatch Synthetics

To start using CloudWatch Synthetics with your VPC, create an interface VPC endpoint for CloudWatch Synthetics. The service name to choose is `com.amazonaws.region.synthetics`. For more information, see Creating an interface endpoint in the *Amazon VPC User Guide*.

You do not need to change the settings for CloudWatch Synthetics. CloudWatch Synthetics communicates with other Amazon services using either public endpoints or private interface VPC endpoints, whichever are in use. For example, if you create an interface VPC endpoint for CloudWatch Synthetics, and you already have an interface endpoint for Amazon S3, CloudWatch Synthetics begins communicating with Amazon S3 through the interface VPC endpoint by default.

Controlling access to your CloudWatch Synthetics VPC endpoint

A VPC endpoint policy is an IAM resource policy that you attach to an endpoint when you create or modify the endpoint. If you don't attach a policy when you create an endpoint, we attach a default policy for you that allows full access to the service. An endpoint policy doesn't override or replace user policies or service-specific policies. It's a separate policy for controlling access from the endpoint to the specified service.

Endpoint policies affect canaries that are managed privately by VPC. They are not needed for canaries that run on private subnets.

Endpoint policies must be written in JSON format.

For more information, see [Controlling access to services with VPC endpoints]() in the *Amazon VPC User Guide*.

The following is an example of an endpoint policy for CloudWatch Synthetics. This policy enables users connecting to CloudWatch Synthetics through the VPC to view information about canaries and their runs, but not to create, modify, or delete canaries.

```
{
    "Statement": [
        {
            "Action": [
                "synthetics:DescribeCanaries",
                "synthetics:GetCanaryRuns"
            ],
            "Effect": "Allow",
            "Resource": "*",
            "Principal": "*"
        }
    ]
}
```

To edit the VPC endpoint policy for CloudWatch Synthetics

1. Open the Amazon VPC console at https://console.amazonaws.cn/vpc/.
2. In the navigation pane, choose **Endpoints**.
3. If you have not already created the endpoint for CloudWatch Synthetics, choose **Create endpoint**. Select **com.amazonaws.*region*.synthetics** and then choose **Create endpoint**.
4. Select the **com.amazonaws.*region*.synthetics** endpoint, and then choose the **Policy** tab.
5. Choose **Edit policy**, and then make your changes.

CloudWatch Network Monitoring feature VPC endpoints

CloudWatch Network Monitoring includes the following features: Network Flow Monitor, Internet Monitor, and Network Synthetic Monitor. These features each support VPC endpoints in the Amazon Regions where the Network Monitoring feature is supported.

To see a list of supported Regions for each Network Monitoring feature, see the following topics:

- **Network Flow Monitor:** Supported Amazon Web Services Regions for Network Flow Monitor
- **Internet Monitor:** Supported Amazon Web Services Regions for Internet Monitor
- **Network Synthetic Monitor:** Supported Amazon Web Services Regions for Network Synthetic Monitor

Creating a VPC endpoint for a CloudWatch Network Monitoring feature

To start using CloudWatch Network Monitoring features with your VPC, create an interface VPC endpoint for the feature that you want to use. For Network Monitoring, the following service names are available:

- `com.amazonaws.region.networkflowmonitor`
- `com.amazonaws.region.networkflowmonitorreports`
- `com.amazonaws.region.internetmonitor`
- `com.amazonaws.region.internetmonitor-fips`
- `com.amazonaws.region.networkmonitor`

For more information, see Creating an interface endpoint in the *Amazon VPC User Guide*.

You don't need to change the settings for Network Monitoring services. Network Monitoring services communicate with other Amazon services using either public endpoints or private interface VPC endpoints, whichever are being used. For example, if you create an interface VPC endpoint for a Network Monitoring service, and you already have metrics flowing to the service from resources located on your VPC, the metrics begin flowing through the interface VPC endpoint by default.

Controlling access to your CloudWatch Network Monitoring feature VPC endpoints

A VPC endpoint policy is an IAM resource policy that you attach to an endpoint when you create or modify the endpoint. An endpoint policy doesn't override or replace user policies or service-specific policies. It's a separate policy for controlling access from the endpoint to the specified service.

If you don't attach a policy when you create an endpoint, Amazon VPC attaches a default policy for you that allows full access and does not restrict access to a specific service. For additional security, you can attach a policy to the endpoint to specifically limit access to the feature. For example, for

Internet Monitor, you could allow full access to just Internet Monitor by attaching the Amazon managed policy that enables full access to the feature, CloudWatchInternetMonitorFullAccess. Or, you can further limit permissions to just specific actions for the endpoint.

For more information, see Controlling access to services with VPC endpoints in the *Amazon VPC User Guide*.

The following is an example of an endpoint policy that you could create for Network Flow Monitor to limit actions for the endpoint. This policy allows requests to Network Flow Monitor through the VPC to use only the `Publish` action, which enables requests to publish metrics to Network Flow Monitor backend ingestion.

```
{
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "networkflowmonitor:Publish",
            "Resource": "*"
        }
    ]
}
```

If you want to use a specific VPC endpoint policy with an interface VPC endpoint for a Network Monitoring feature, use steps similar to the following example for adding a policy for Network Flow Monitor.

To edit a VPC endpoint policy for Network Flow Monitor

1. Open the Amazon VPC console at https://console.amazonaws.cn/vpc/.
2. In the navigation pane, choose **Endpoints**.
3. If you have not already created the endpoint for Internet Monitor, choose **Create endpoint**.
4. Select **com.amazonaws.*region*.networkflowmonitor**, and then choose **Create endpoint**.
5. Select the **com.amazonaws.*region*.networkflowmonitor** endpoint, and then choose the **Policy** tab.
6. Choose **Edit policy**, and then make your changes.

Security considerations for Synthetics canaries

The following sections explain security issues that you should consider when creating and running canaries in Synthetics.

Use secure connections

Because canary code and the results from canary test runs can contain sensitive information, do not have your canary connect to endpoints over unencrypted connections. Always use encrypted connections, such as those that begin with `https://`.

Canary naming considerations

The Amazon Resource Name (ARN) of a canary is included in the user-agent header as a part of outbound calls made from the Puppeteer-driven Chromium browser that is included as a part of the CloudWatch Synthetics wrapper library. This helps identify CloudWatch Synthetics canary traffic and relate it back to the canaries that are making calls.

The canary ARN includes the canary name. Choose canary names that do not reveal proprietary information.

Additionally, be sure to point your canaries only at websites and endpoints that you control.

Secrets and sensitive information in canary code

If you pass your canary code directly into the canary using a zip file, the contents of the script can be seen in Amazon CloudTrail logs.

If you have sensitive information or secrets (such as access keys or database credentials) in a canary script, we strongly recommend that you store the script as a versioned object in Amazon S3 and pass the Amazon S3 location into the canary, instead of passing the canary code by a zip file.

If you do use a zip file to pass the canary script, we strongly recommend that you don't include secrets or sensitive information in your canary source code. For more information about how to use Amazon Secrets Manager to help keep your secrets safe, see [What is Amazon Secrets Manager?](https://docs.aws.amazon.com/secretsmanager/latest/userguide/intro.html).

Permissions considerations

We recommend that you restrict access to resources that are created or used by CloudWatch Synthetics. Use tight permissions on the Amazon S3 buckets where canaries store test run results and other artifacts, such as logs and screenshots.

Similarly, keep tight permissions on the locations where your canary source code is stored, so that no user accidentally or maliciously deletes the Lambda layers or Lambda functions used for the canary.

To help make sure you run the canary code you intend, you can use object versioning on the Amazon S3 bucket where your canary code is stored. Then when you specify this code to run as a canary, you can include the object `versionId` as part of the path, as in the following examples.

```
https://bucket.s3.amazonaws.com/path/object.zip?versionId=version-id
https://s3.amazonaws.com/bucket/path/object.zip?versionId=version-id
https://bucket.s3-region.amazonaws.com/path/object.zip?versionId=version-id
```

Stack traces and exception messages

By default, CloudWatch Synthetics canaries capture any exception thrown by your canary script, no matter whether the script is custom or is from a blueprint. CloudWatch Synthetics logs both the exception message and the stack trace to three locations:

- Back into the CloudWatch Synthetics service to speed up debugging when you describe test runs
- Into CloudWatch Logs according to the configuration that your Lambda functions are created with
- Into the Synthetics log file, which is a plaintext file that is uploaded to the Amazon S3 location specified by the value you set for the `resultsLocation` of the canary

If you want to send and store less information, you can capture exceptions before they return to the CloudWatch Synthetics wrapper library.

You can also have request URLs in your errors. CloudWatch Synthetics scans for any URLs in the error thrown by your script and redacts restricted URL parameters from them based on the **restrictedUrlParameters** configuration. If you are logging error messages in your script, you can use getSanitizedErrorMessage to redact URLs before logging.

Scope your IAM roles narrowly

We recommend that you do not configure your canary to visit potentially malicious URLs or endpoints. Pointing your Canary to untrusted or unknown websites or endpoints could expose your Lambda function code to malicious user’s scripts. Assuming a malicious website can break out of

Chromium, it could have access to your Lambda code in a similar way to if you connected to it using an internet browser.

Run your Lambda function with an IAM execution role that has scoped-down permissions. This way, if your Lambda function is compromised by a malicious script, it is limited in the actions it can take when running as your canary's Amazon account.

When you use the CloudWatch console to create a canary, it is created with a scoped-down IAM execution role.

Sensitive data redaction

CloudWatch Synthetics captures URLs, status code, failure reason (if any), and headers and bodies of requests and responses. This enables a canary user to understand, monitor, and debug canaries.

The configurations described in the following sections can be set at any point in canary execution. You can also choose to apply different configurations to different synthetics steps.

Request URLs

By default, CloudWatch Synthetics logs request URLs, status codes, and the status reason for each URL in canary logs. Request URLs can also appear in canary execution reports, HAR files, and so on. Your request URL might contain sensitive query parameters, such as access tokens or passwords. You can redact sensitive information from being logged by CloudWatch Synthetics.

To redact sensitive information, set the configuration property **restrictedUrlParameters**. For more information, see SyntheticsConfiguration class. This causes CloudWatch Synthetics to redact URL parameters, including path and query parameter values, based on **restrictedUrlParameters** before logging. If you are logging URLs in your script, you can use getSanitizedUrl(url, stepConfig = null) to redact URLs before logging. For more information, see SyntheticsLogHelper class.

Headers

By default, CloudWatch Synthetics doesn't log request/response headers. For UI canaries, this is the default behavior for canaries using runtime version `syn-nodejs-puppeteer-3.2` and later.

If your headers don't contain sensitive information, you can enable headers in HAR file and HTTP reports by setting the **includeRequestHeaders** and **includeResponseHeaders** properties to `true`. You can enable all headers but choose to restrict values of sensitive header keys. For example, you can choose to only redact `Authorization` headers from artifacts produced by canaries.

Request and response body

By default, CloudWatch Synthetics doesn't log the request/response body in canary logs or reports. This information is particularly useful for API canaries. Synthetics captures all HTTP requests and can show headers, request and response bodies. For more information, see executeHttpStep(stepName, requestOptions, [callback], [stepConfig]). You can choose to enable request/response body by setting the **includeRequestBody** and **includeResponseBody** properties to `true`.

Logging Amazon CloudWatch API and console operations with Amazon CloudTrail

Amazon CloudWatch, CloudWatch Synthetics, CloudWatch RUM, Amazon Q Developer operational investigations, Network Flow Monitor, and Internet Monitor are integrated with Amazon CloudTrail, a service that provides a record of actions taken by a user, role, or an Amazon service. CloudTrail captures API calls made by or on behalf of your Amazon account. The captured calls include calls from the CloudWatch console and code calls to CloudWatch API operations. Using the information collected by CloudTrail, you can determine the request that was made to CloudWatch, the IP address from which the request was made, when it was made, and additional details.

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root user or user credentials.
- Whether the request was made on behalf of an IAM Identity Center user.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another Amazon Web Services service.

CloudTrail is active in your Amazon Web Services account when you create the account and you automatically have access to the CloudTrail **Event history**. The CloudTrail **Event history** provides a viewable, searchable, downloadable, and immutable record of the past 90 days of recorded management events in an Amazon Web Services Region. For more information, see Working with CloudTrail Event history in the *Amazon CloudTrail User Guide*. There are no CloudTrail charges for viewing the **Event history**.

For an ongoing record of events in your Amazon Web Services account past 90 days, create a trail or a CloudTrail Lake event data store.

CloudTrail trails

A *trail* enables CloudTrail to deliver log files to an Amazon S3 bucket. All trails created using the Amazon Web Services Management Console are multi-Region. You can create a single-Region or a multi-Region trail by using the Amazon CLI. Creating a multi-Region trail is recommended because you capture activity in all Amazon Web Services Regions in your account. If you create a single-Region trail, you can view only the events logged in the trail's Amazon Web Services

Region. For more information about trails, see Creating a trail for your Amazon Web Services account and Creating a trail for an organization in the *Amazon CloudTrail User Guide*.

You can deliver one copy of your ongoing management events to your Amazon S3 bucket at no charge from CloudTrail by creating a trail, however, there are Amazon S3 storage charges. For more information about CloudTrail pricing, see Amazon CloudTrail Pricing. For information about Amazon S3 pricing, see Amazon S3 Pricing.

CloudTrail Lake event data stores

CloudTrail Lake lets you run SQL-based queries on your events. CloudTrail Lake converts existing events in row-based JSON format to Apache ORC format. ORC is a columnar storage format that is optimized for fast retrieval of data. Events are aggregated into *event data stores*, which are immutable collections of events based on criteria that you select by applying advanced event selectors. The selectors that you apply to an event data store control which events persist and are available for you to query. For more information about CloudTrail Lake, see Working with Amazon CloudTrail Lake in the *Amazon CloudTrail User Guide*.

CloudTrail Lake event data stores and queries incur costs. When you create an event data store, you choose the pricing option you want to use for the event data store. The pricing option determines the cost for ingesting and storing events, and the default and maximum retention period for the event data store. For more information about CloudTrail pricing, see Amazon CloudTrail Pricing.

> **Note**
>
> For information about CloudWatch Logs API calls that are logged in CloudTrail, see CloudWatch Logs information in CloudTrail.

Topics

- CloudWatch information in CloudTrail
- CloudWatch data events in CloudTrail
- Query generation information in CloudTrail
- Amazon Q Developer operational investigations events in CloudTrail
- Network Flow Monitor in CloudTrail
- Network Flow Monitor data plane events in CloudTrail

- Internet Monitor in CloudTrail
- CloudWatch Synthetics information in CloudTrail
- CloudWatch RUM information in CloudTrail
- CloudWatch RUM data plane events in CloudTrail

CloudWatch information in CloudTrail

CloudWatch supports logging the following actions as events in CloudTrail log files:

- DeleteAlarms
- DeleteAnomalyDetector
- DeleteDashboards
- DescribeAlarmHistory
- DescribeAlarms
- DescribeAlarmsForMetric
- DescribeAnomalyDetectors
- DisableAlarmActions
- EnableAlarmActions
- GetDashboard
- ListDashboards
- PutAnomalyDetector
- PutDashboard
- PutMetricAlarm
- SetAlarmState
- TagResource
- UntagResource

Example: CloudWatch log file entries

The following example shows a CloudTrail log entry that demonstrates the `PutMetricAlarm` action.

```
{
    "Records": [{
        "eventVersion": "1.01",
        "userIdentity": {
            "type": "Root",
            "principalId": "EX_PRINCIPAL_ID",
            "arn": "arn:aws:iam::123456789012:root",
            "accountId": "123456789012",
            "accessKeyId": "EXAMPLE_KEY_ID"
        },
        "eventTime": "2014-03-23T21:50:34Z",
        "eventSource": "monitoring.amazonaws.com",
        "eventName": "PutMetricAlarm",
        "awsRegion": "us-east-1",
        "sourceIPAddress": "127.0.0.1",
        "userAgent": "aws-sdk-ruby2/2.0.0.rc4 ruby/1.9.3 x86_64-linux Seahorse/0.1.0",
        "requestParameters": {
            "threshold": 50.0,
            "period": 60,
            "metricName": "CloudTrail Test",
            "evaluationPeriods": 3,
            "comparisonOperator": "GreaterThanThreshold",
            "namespace": "AWS/CloudWatch",
            "alarmName": "CloudTrail Test Alarm",
            "statistic": "Sum"
        },
        "responseElements": null,
        "requestID": "29184022-b2d5-11e3-a63d-9b463e6d0ff0",
        "eventID": "b096d5b7-dcf2-4399-998b-5a53eca76a27"
    },
    ..additional entries
  ]
  }
```


The following log file entry shows that a user called the CloudWatch Events `PutRule` action.


```
{
         "eventVersion":"1.03",
         "userIdentity":{
            "type":"Root",
            "principalId":"123456789012",
            "arn":"arn:aws:iam::123456789012:root",
            "accountId":"123456789012",
```

```
            "accessKeyId":"AKIAIOSFODNN7EXAMPLE",
            "sessionContext":{
               "attributes":{
                  "mfaAuthenticated":"false",
                  "creationDate":"2015-11-17T23:56:15Z"
               }
            }
         },
         "eventTime":"2015-11-18T00:11:28Z",
         "eventSource":"events.amazonaws.com",
         "eventName":"PutRule",
         "awsRegion":"us-east-1",
         "sourceIPAddress":"AWS Internal",
         "userAgent":"AWS CloudWatch Console",
         "requestParameters":{
            "description":"",
            "name":"cttest2",
            "state":"ENABLED",
            "eventPattern":"{\"source\":[\"aws.ec2\"],\"detail-type\":[\"EC2 Instance
 State-change Notification\"]}",
            "scheduleExpression":""
         },
         "responseElements":{
            "ruleArn":"arn:aws:events:us-east-1:123456789012:rule/cttest2"
         },
         "requestID":"e9caf887-8d88-11e5-a331-3332aa445952",
         "eventID":"49d14f36-6450-44a5-a501-b0fdcdfaeb98",
         "eventType":"AwsApiCall",
         "apiVersion":"2015-10-07",
         "recipientAccountId":"123456789012"
}
```


The following log file entry shows that a user called the CloudWatch Logs `CreateExportTask` action.


```
{
        "eventVersion": "1.03",
        "userIdentity": {
            "type": "IAMUser",
            "principalId": "EX_PRINCIPAL_ID",
            "arn": "arn:aws:iam::123456789012:user/someuser",
            "accountId": "123456789012",
            "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
```

```
            "userName": "someuser"
        },
        "eventTime": "2016-02-08T06:35:14Z",
        "eventSource": "logs.amazonaws.com",
        "eventName": "CreateExportTask",
        "awsRegion": "us-east-1",
        "sourceIPAddress": "127.0.0.1",
        "userAgent": "aws-sdk-ruby2/2.0.0.rc4 ruby/1.9.3 x86_64-linux Seahorse/0.1.0",
        "requestParameters": {
            "destination": "yourdestination",
            "logGroupName": "yourloggroup",
            "to": 123456789012,
            "from": 0,
            "taskName": "yourtask"
        },
        "responseElements": {
            "taskId": "15e5e534-9548-44ab-a221-64d9d2b27b9b"
        },
        "requestID": "1cd74c1c-ce2e-12e6-99a9-8dbb26bd06c9",
        "eventID": "fd072859-bd7c-4865-9e76-8e364e89307c",
        "eventType": "AwsApiCall",
        "apiVersion": "20140328",
        "recipientAccountId": "123456789012"
}
```

CloudWatch data events in CloudTrail

CloudTrail can capture API activities related to the CloudWatch data plane operations GetMetricData and GetMetricWidgetImage.

Data events, also known as data plane operations, give you insight into the resource operations performed on or within a resource. Data events are often high-volume activities.

By default, CloudTrail doesn’t log data events. The CloudTrail **Event history** doesn't record data events.

Additional charges apply for data events. For more information about CloudTrail pricing, see Amazon CloudTrail Pricing.

You can log data events for the CloudWatch resource types by using the CloudTrail console, Amazon CLI, or CloudTrail API operations. For more information about how to log data events, see

Logging data events with the Amazon Web Services Management Console and Logging data events with the Amazon Command Line Interface in the *Amazon CloudTrail User Guide*.

Data plane events can be filtered by resource type. Because there are additional costs for using data events in CloudTrail, filtering by resource allows you to have more control over what you log and pay for.

Using the information that CloudTrail collects, you can identify a specific request to the CloudWatch GetMetricData or GetMetricWidgetImage APIs, the IP address of the requester, the requester's identity, and the date and time of the request. Logging the **GetMetricData** and **GetMetricWidgetImage** APIs using CloudTrail helps you enable operational and risk auditing, governance, and compliance of your Amazon account.

Note

When you view the **GetMetricData** events in CloudTrail, you might see more calls than the calls that you initiated. This is because CloudWatch logs events to CloudTrail for **GetMetricData** actions that are initiated by internal components. For example, you'll see **GetMetricData** calls initiated by CloudWatch dashboards to refresh widget data, and **GetMetricData** calls initiated by a monitoring account to retrieve data from a source account, in cross-account observability. These internally-initiated calls don't incur CloudWatch charges, but they do count toward the number of events logged in CloudTrail, and CloudTrail charges according to the number of events logged.

The following is an example of a CloudTrail event for a **GetMetricData** operation.


```
{
    "eventVersion": "1.09",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AIDA2NYTR2EPCTNY7AF3L",
        "arn": "arn:aws:iam::111122223333:user/admin",
        "accountId": "111122223333",
        "accessKeyId": "EXAMPLE1234567890",
        "userName": "admin"
    },
    "eventTime": "2024-05-08T16:20:34Z",
    "eventSource": "monitoring.amazonaws.com",
    "eventName": "GetMetricData",
```

```
    "awsRegion": "us-east-1",
    "sourceIPAddress": "99.45.3.7",
    "userAgent": "aws-cli/2.13.23 Python/3.11.5 Darwin/23.4.0 exe/x86_64 prompt/off
 command/cloudwatch.get-metric-data",
    "requestParameters": {
        "metricDataQueries": [{
                "id": "e1",
                "expression": "m1 / m2",
                "label": "ErrorRate"
            },
            {
                "id": "m1",
                "metricStat": {
                    "metric": {
                        "namespace": "CWAgent",
                        "metricName": "disk_used_percent",
                        "dimensions": [{
                            "name": "LoadBalancerName",
                            "value": "EXAMPLE4623a5cb6a7384c5229"
                        }]
                    },
                    "period": 300,
                    "stat": "Sum",
                    "unit": "Count"
                },
                "returnData": false
            },
            {
                "id": "m2",
                "metricStat": {
                    "metric": {
                        "namespace": "CWAgent",
                        "metricName": "disk_used_percent",
                        "dimensions": [{
                            "name": "LoadBalancerName",
                            "value": "EXAMPLE4623a5cb6a7384c5229"
                        }]
                    },
                    "period": 300,
                    "stat": "Sum"
                },
                "returnData": true
            }
        ],
```

```
        "startTime": "Apr 19, 2024, 4:00:00 AM",
        "endTime": "May 8, 2024, 4:30:00 AM"
    },
    "responseElements": null,
    "requestID": "EXAMPLE-57ac-47d5-938c-f5917c6799d5",
    "eventID": "EXAMPLE-211c-404b-b13d-36d93c8b4fbf",
    "readOnly": true,
    "resources": [{
        "type": "AWS::CloudWatch::Metric"
    }],
    "eventType": "AwsApiCall",
    "managementEvent": false,
    "recipientAccountId": "111122223333",
    "eventCategory": "Data",
    "tlsDetails": {
        "tlsVersion": "TLSv1.3",
        "cipherSuite": "TLS_AES_128_GCM_SHA256",
        "clientProvidedHostHeader": "monitoring.us-east-1.amazonaws.com"
    }
}
```

Query generation information in CloudTrail

CloudTrail logging for Query generator console events is also supported. Query generator is currently supported for CloudWatch Metric Insights and CloudWatch Logs Insights. In these CloudTrail events, the `eventSource` is `monitoring.amazonaws.com`.

The following example shows a CloudTrail log entry that demonstrates the **GenerateQuery** action in CloudWatch Metrics Insights.

```
{
    "eventVersion": "1.09",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::123456789012:assumed-role/role_name",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "EX_PRINCIPAL_ID",
```

```
                "arn": "arn:aws:iam::111222333444:role/Administrator",
                "accountId": "123456789012",
                "userName": "SAMPLE_NAME"
            },
            "attributes": {
                "creationDate": "2020-04-08T21:43:24Z",
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2020-04-08T23:06:30Z",
    "eventSource": "monitoring.amazonaws.com",
    "eventName": "GenerateQuery",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "127.0.0.1",
    "userAgent": "exampleUserAgent",
    "requestParameters": {
        "query_ask": "***",
        "query_type": "MetricsInsights",
        "metrics_insights": {
            "aws_namespaces": [
                "AWS/S3",
                "AWS/Lambda",
                "AWS/DynamoDB"
            ]
        },
        "include_description": true
    },
    "responseElements": null,
    "requestID": "2f56318c-cfbd-4b60-9d93-1234567890",
    "eventID": "52723fd9-4a54-478c-ac55-1234567890",
    "readOnly": true,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```

Amazon Q Developer operational investigations events in CloudTrail

Amazon Q Developer operational investigations supports logging the following actions as events in CloudTrail log files.

- CreateInvestigation
- GetInvestigation
- UpdateInvestigation
- DeleteInvestigation
- ListInvestigations
- CreateInvestigationGroup
- GetInvestigationGroup
- UpdateInvestigationGroup
- DeleteInvestigationGroup
- ListInvestigationsGroup
- PutInvestigationGroupPolicy
- DeleteInvestigationGroupPolicy
- ListTagsForResource
- GetInvestigationGroupPolicy
- TagResource
- UntagResource
- CreateInvestigationEvent
- GetInvestigationEvent
- UpdateInvestigationEvent
- ListInvestigationEvents
- CreateInvestigationResource
- GetInvestigationResource

Example: Amazon Q Developer operational investigations log file entries

The following example shows a Amazon Q Developer operational investigations log entry that demonstrates the `CreateInvestigationGroup` action.


```
{
 "eventVersion": "1.09",
 "userIdentity": {
  "type": "AssumedRole",
  "principalId": "EX_PRINCIPAL_ID",
  "arn": "arn:aws:iam::123456789012:assumed-role/role_name",
  "accountId": "123456789012",
  "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
  "sessionContext": {
   "sessionIssuer": {
    "type": "Role",
    "principalId": "EX_PRINCIPAL_ID",
    "arn": "arn:aws:iam::123456789012:role/role_name",
    "accountId": "123456789012",
    "userName": "SAMPLE_NAME"
   },
   "attributes": {
    "creationDate": "2024-10-30T18:42:05Z",
    "mfaAuthenticated": "false"
   }
  }
 },
 "eventTime": "2024-10-30T18:48:26Z",
 "eventSource": "aiops.amazonaws.com",
 "eventName": "CreateInvestigationGroup",
 "awsRegion": "us-east-1",
 "sourceIPAddress": "127.0.0.1",
 "userAgent": "exampleUserAgent",
 "requestParameters": {
  "name": "exampleName",
  "roleArn": "arn:aws:iam::123456789012:role/role_name" },
 "responseElements": {
  "arn": "arn:aws:aiops:us-east-1:123456789012:investigation-group/021345abcdef67890"
 },
 "requestId": "e9caf887-8d88-11e5-a331-3332aa445952",
 "requestId": "49d14f36-6450-44a5-a501-b0fdcdfaeb98",
 "readOnly": false,
```

```
 "eventType": "AwsApiCall",
 "managementEvent": true,
 "recipientAccountId": "123456789012",
 "eventCategory": "Management"
}
```


The following example shows a Amazon Q Developer operational investigations log entry that demonstrates the `CreateInvestigationEvent` action.


```
{
  "eventVersion": "1.09",
  "userIdentity": {
   "type": "AssumedRole",
   "principalId": "EX_PRINCIPAL_ID",
   "arn": "arn:aws:sts::123456789012:assumed-role/role_name",
   "accountId": "123456789012",
   "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
   "sessionContext": {
    "sessionIssuer": {
     "type": "Role",
     "principalId": "EX_PRINCIPAL_ID",
     "arn": "arn:aws:iam::123456789012:role/role_name",
     "accountId": "123456789012",
     "userName": "SAMPLE_NAME"
    },
    "attributes": {
     "creationDate": "2024-10-30T16:17:49Z",
     "mfaAuthenticated": "false"
    }
   }
  },
  "eventTime": "2024-10-30T16:35:34Z",
  "eventSource": "aiops.amazonaws.com",
  "eventName": "CreateInvestigationEvent",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "127.0.0.1",
  "userAgent": "exampleUserAgent",
  "requestParameters": {
   "identifier": "arn:aws:aiops:us-east-1:123456789012:investigation-
group/021345abcdef67890",
   "investigationId": "bcdef01234567890",
   "clientToken": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
   "type": "METRIC_OBSERVATION",
```

```
    "body": "***"
   },
   "responseElements": {
    "investigationGroupArn": "arn:aws:aiops:us-east-1:123456789012:investigation-
group/021345abcdef67890",
    "investigationId": "bcdef01234567890",
    "investigationEventId": "14567890abcdef0g"
   },
   "requestId": "e9caf887-8d88-11e5-a331-3332aa445952",
   "eventId": "49d14f36-6450-44a5-a501-b0fdcdfaeb98",
   "readOnly": false,
   "resources": [{
    "accountId": "123456789012",
    "type": "AWS::AIOps::InvestigationGroup",
    "ARN": "arn:aws:aiops:us-east-1:123456789012:investigation-group/021345abcdef67890"
   }],
   "eventType": "AwsApiCall",
   "managementEvent": false,
   "recipientAccountId": "123456789012",
   "eventCategory": "Data"
  }
```


The following example shows a Amazon Q Developer operational investigations log entry that demonstrates the `UpdateInvestigationEvent` action.


```
{
  "eventVersion": "1.09",
  "userIdentity": {
   "type": "AssumedRole",
   "principalId": "EX_PRINCIPAL_ID",
   "arn": "arn:aws:sts::123456789012:assumed-role/role_name",
   "accountId": "123456789012",
   "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
   "sessionContext": {
    "sessionIssuer": {
     "type": "Role",
     "principalId": "EX_PRINCIPAL_ID",
     "arn": "arn:aws:iam::123456789012:role/role_name",
     "accountId": "123456789012",
     "userName": "SAMPLE_NAME"
    },
    "attributes": {
     "creationDate": "2024-10-30T16:17:49Z",
```

```
    "mfaAuthenticated": "false"
   }
  }
 },
 "eventTime": "2024-10-30T16:24:48Z",
 "eventSource": "aiops.amazonaws.com",
 "eventName": "UpdateInvestigationEvent",
 "awsRegion": "us-east-1",
 "sourceIPAddress": "127.0.0.1",
 "userAgent": "exampleUserAgent",
 "requestParameters": {
  "identifier": "arn:aws:aiops:us-east-1:123456789012:investigation-
group/021345abcdef67890",
  "investigationId": "bcdef01234567890",
  "investigationEventId": "14567890abcdef0g",
  "comment": "***"
 },
 "responseElements": null,
 "requestId": "e9caf887-8d88-11e5-a331-3332aa445952",
 "eventId": "49d14f36-6450-44a5-a501-b0fdcdfaeb98",
 "readOnly": false,
 "resources": [{
  "accountId": "123456789012",
  "type": "AWS::AIOps::InvestigationGroup",
  "ARN": "arn:aws:aiops:us-east-1:123456789012:investigation-group/021345abcdef67890"
 }],
 "eventType": "AwsApiCall",
 "managementEvent": false,
 "recipientAccountId": "123456789012",
 "eventCategory": "Data"
}
```


Network Flow Monitor in CloudTrail

Network Flow Monitor supports logging the following actions as events in CloudTrail log files.

- CreateMonitor
- GetMonitor
- DeleteMonitor
- ListMonitors
- UpdateMonitor

- CreateScope
- GetScope
- ListScopes
- DeleteScope
- UpdateScope

Example: Network Flow Monitor log file entry

The following example shows a Network Flow Monitor CloudTrail log file entry that demonstrates the `CreateMonitor` action.


```
{
    "eventVersion": "1.09",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::000000000000:assumed-role/role_name",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:iam::000000000000:role/Admin",
                "accountId": "123456789012",
                "userName": "SAMPLE_NAME"
            },
            "attributes": {
                "creationDate": "2024-11-03T15:43:27Z",
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2024-11-03T15:58:11Z",
    "eventSource": "networkflowmonitor.amazonaws.com",
    "eventName": "CreateMonitor",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)",
    "requestParameters": {
```

```
        "MonitorName": "TestMonitor",
        "ClientToken": "33551db7-1618-4aab-cdef-EXAMPLE33333",
        "LocalResources": [
            {
                "Type": "AWS::EC2::Subnet",
                "Identifier": "subnet-cdef-EXAMPLEbbbbb"
            },
            {
                "Type": "AWS::EC2::Subnet",
                "Identifier": "subnet-cdef-EXAMPLEccccc"
            },
            {
                "Type": "AWS::EC2::Subnet",
                "Identifier": "subnet-cdef-EXAMPLEddddd"
            },
            {
                "Type": "AWS::EC2::Subnet",
                "Identifier": "subnet-cdef-EXAMPLEeeeee"
            },
            {
                "Type": "AWS::EC2::Subnet",
                "Identifier": "subnet-cdef-EXAMPLEfffff"
            },
            {
                "Type": "AWS::EC2::Subnet",
                "Identifier": "subnet-cdef-EXAMPLEggggg"
            }
        ]
    },
    "responseElements": {
        "Access-Control-Expose-Headers": "*",
        "MonitorArn": "arn:aws:networkflowmonitor:us-east-1:000000000000:monitor/
TestMonitor",
        "MonitorName": "TestMonitor",
        "MonitorStatus": "ACTIVE"
    },
    "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
    "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEbbbbb",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
```

```
}
```



```
{
        "eventVersion": "1.08",
        "userIdentity": {
            "type": "AssumedRole",
            "principalId": "EX_PRINCIPAL_ID",
            "arn": "arn:aws:iam::000000000000:assumed-role/role_name",
            "accountId":"123456789012",
            "accessKeyId":"AKIAIOSFODNN7EXAMPLE",
            "sessionContext": {
                "sessionIssuer": {
                "type": "Role",
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:iam::000000000000:role/Admin",
                "accountId":"123456789012",
                "userName": "SAMPLE_NAME"
                },
                "webIdFederationData": {},
                "attributes": {
                    "creationDate": "2022-10-11T17:25:41Z",
                    "mfaAuthenticated": "false"
                }
            }
        },
        "eventTime": "2022-10-11T17:30:18Z",
        "eventSource": "networkflowmonitor.amazonaws.com",
        "eventName": "ListMonitors",
        "awsRegion": "us-east-2",
        "sourceIPAddress": "192.0.2.0",
        "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)",
        "requestParameters": null,
        "responseElements": null,
        "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEbbbbb",
        "readOnly": true,
        "eventType": "AwsApiCall",
        "managementEvent": true,
        "recipientAccountId": "111122223333",
        "eventCategory": "Management"
    }
```

Network Flow Monitor data plane events in CloudTrail

CloudTrail can capture API activities related to the CloudWatch-NetworkFlowMonitor data plane operations.

Data events, also known as data plane operations, give you insight into the resource operations performed on or within a resource. Data events are often high-volume activities.

To enable logging of Network Flow Monitor data events in CloudTrail files, you'll need to enable the logging of data plane API activity in CloudTrail. See Logging data events for trails for more information.

Data plane events can be filtered by resource type. Because there are additional costs for using data events in CloudTrail, filtering by resource allows you to have more control over what you log and pay for.

Using the information that CloudTrail collects, you can identify a specific request to the CloudWatch-NetworkFlowMonitor data plane APIs, the IP address of the requester, the requester's identity, and the date and time of the request. Logging the data plane APIs using CloudTrail can help you with operational and risk auditing, governance, and compliance of your Amazon account.

The following are data plane APIs in Network Flow Monitor.

- StartQueryWorkloadInsightsTopContributors
- StopQueryWorkloadInsightsTop Contributors
- GetQueryStatusWorkloadInsightsTopContributors
- GetQueryResultsWorkloadInsightsTopContributors
- StartQueryWorkloadInsightsTopContributorsData
- StopQueryWorkloadInsightsTopContributorsData
- GetQueryStatusWorkloadInsightsTopContributorsData
- GetQueryResultsWorkloadInsightsTopContributorsData
- StartQueryMonitorsTopContributors
- StopQueryMonitorsTopContributors
- GetQueryStatusMonitorsTopContributors
- GetQueryResultsMonitorsTopContributors

The following example shows a CloudTrail log entry that demonstrates the GetQueryResultsMonitorsTopContributors action.


```
{
  "eventVersion": "1.09",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "EX_PRINCIPAL_ID",
    "arn": "arn:aws:iam::000000000000:assumed-role/role_name",
    "accountId": "123456789012",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "sessionContext": {
      "sessionIssuer": {
        "type": "Role",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::000000000000:role/Admin",
        "accountId": "123456789012",
         "userName": "SAMPLE_NAME"
      },
      "attributes": {
      "creationDate": "2024-11-03T15:43:27Z",
      "mfaAuthenticated": "false"
      }
    }
},
  "eventTime": "2024-11-15T14:08:04Z",
  "eventSource": "networkflowmonitor.amazonaws.com",
  "eventName": "GetQueryResultsMonitorTopContributors",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "192.0.2.0",
  "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)",
  "errorCode": "AccessDenied",
  "requestParameters": {
    "QueryId": "a1b2c3d4-5678-90ab-cdef-EXAMPLEQuery,
    "MaxResults": "20",
    "MonitorName": "TestMonitor"
  },
  "responseElements": null,
  "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
  "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEbbbbb",
  "readOnly": true,
  "resources": [
    {
```

```
      "accountId": "123456789012",
      "type": "AWS::NetworkFlowMonitor::Monitor",
      "ARN": "arn:aws:networkflowmonitor:us-east-1:123456789012:monitor/TestMonitor"
    }
  ],
  "eventType": "AwsApiCall",
  "managementEvent": false,
  "recipientAccountId": "000000000000",
  "eventCategory": "Data"
}
```

The following example shows a CloudTrail log entry that demonstrates the GetQueryResultsWorkloadInsightsTopContributors action.

```
{
  "eventVersion": "1.09",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "EX_PRINCIPAL_ID",
    "arn": "arn:aws:iam::000000000000:assumed-role/role_name",
    "accountId": "123456789012",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "sessionContext": {
      "sessionIssuer": {
        "type": "Role",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::000000000000:role/Admin",
        "accountId": "123456789012",
         "userName": "SAMPLE_NAME"
      },
      "attributes": {
      "creationDate": "2024-11-03T15:43:27Z",
      "mfaAuthenticated": "false"
      }
    }
},
  "eventTime": "2024-11-15T14:08:04Z",
  "eventSource": "networkflowmonitor.amazonaws.com",
  "eventName": "GetQueryResultsWorkloadInsightsTopContributorsData",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "192.0.2.0",
  "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)",
  "errorCode": "AccessDenied",
```

```
  "requestParameters": {
    "QueryId": "a1b2c3d4-5678-90ab-cdef-EXAMPLEQuery",
    "ScopeId": "a1b2c3d4-5678-90ab-cdef-EXAMPLEScope"
  },
  "responseElements": null,
  "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
  "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEbbbbb",
  "readOnly": true,
  "resources": [
    {
      "accountId": "496383180932",
      "type": "AWS::NetworkFlowMonitor::Scope",
      "ARN": "arn:aws:networkflowmonitor:us-east-1:123456789012:scope/
a1b2c3d4-5678-90ab-cdef-EXAMPLEScope"
    }
  ],
  "eventType": "AwsApiCall",
  "managementEvent": false,
  "recipientAccountId": "000000000000",
  "eventCategory": "Data"
}
```


Internet Monitor in CloudTrail

Internet Monitor supports logging the following actions as events in CloudTrail log files.

- CreateMonitor
- DeleteMonitor
- GetHealthEvent
- GetMonitor
- GetQueryResults
- GetQueryStatus
- ListHealthEvents
- ListMonitors
- ListTagsForResource
- StartQuery
- StopQuery
- UpdateMonitor

Example: Internet Monitor log file entries

The following example shows a CloudTrail Internet Monitor log entry that demonstrates the `ListMonitors` action.


```
{
        "eventVersion": "1.08",
        "userIdentity": {
            "type": "AssumedRole",
            "principalId": "EX_PRINCIPAL_ID",
            "arn": "arn:aws:iam::000000000000:assumed-role/role_name",
            "accountId":"123456789012",
            "accessKeyId":"AKIAIOSFODNN7EXAMPLE",
            "sessionContext": {
                "sessionIssuer": {
                "type": "Role",
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:iam::000000000000:role/Admin",
                "accountId":"123456789012",
                "userName": "SAMPLE_NAME"
                },
                "webIdFederationData": {},
                "attributes": {
                    "creationDate": "2022-10-11T17:25:41Z",
                    "mfaAuthenticated": "false"
                }
            }
        },
        "eventTime": "2022-10-11T17:30:18Z",
        "eventSource": "internetmonitor.amazonaws.com",
        "eventName": "ListMonitors",
        "awsRegion": "us-east-2",
        "sourceIPAddress": "192.0.2.0",
        "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)",
        "requestParameters": null,
        "responseElements": null,
        "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEbbbbb",
        "readOnly": true,
        "eventType": "AwsApiCall",
        "managementEvent": true,
        "recipientAccountId": "111122223333",
        "eventCategory": "Management"
```

```
    }
```

The following example shows a CloudTrail Internet Monitor log entry that demonstrates the `CreateMonitor` action.


```
{
        "eventVersion": "1.08",
        "userIdentity": {
            "type": "AssumedRole",
            "principalId": "EX_PRINCIPAL_ID",
            "arn": "arn:aws:iam::000000000000:assumed-role/role_name",
            "accountId":"123456789012",
            "accessKeyId":"AKIAIOSFODNN7EXAMPLE",
            "sessionContext": {
                "sessionIssuer": {
                "type": "Role",
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:iam::000000000000:role/Admin",
                "accountId":"123456789012",
                "userName": "SAMPLE_NAME"
                },
                "webIdFederationData": {},
                "attributes": {
                    "creationDate": "2022-10-11T17:25:41Z",
                    "mfaAuthenticated": "false"
                }
            }
        },
        "eventTime": "2022-10-11T17:30:08Z",
        "eventSource": "internetmonitor.amazonaws.com",
        "eventName": "CreateMonitor",
        "awsRegion": "us-east-2",
        "sourceIPAddress": "192.0.2.0",
        "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)",
        "requestParameters": {
            "MonitorName": "TestMonitor",
            "Resources": ["arn:aws:ec2:us-east-2:444455556666:vpc/vpc-febc0b95"],
            "ClientToken": "a1b2c3d4-5678-90ab-cdef-EXAMPLE33333"
        },
        "responseElements": {
            "Arn": "arn:aws:internetmonitor:us-east-2:444455556666:monitor/ct-
onboarding-test",
            "Status": "PENDING"
```

```
        },
        "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEbbbbb",
        "readOnly": false,
        "eventType": "AwsApiCall",
        "managementEvent": true,
        "recipientAccountId": "111122223333",
        "eventCategory": "Management"
    }
```


CloudWatch Synthetics information in CloudTrail

CloudWatch Synthetics supports logging the following actions as events in CloudTrail log files:

- CreateCanary
- DeleteCanary
- DescribeCanaries
- DescribeCanariesLastRun
- DescribeRuntimeVersions
- GetCanary
- GetCanaryRuns
- ListTagsForResource
- StartCanary
- StopCanary
- TagResource
- UntagResource
- UpdateCanary

Example: CloudWatch Synthetics log file entries

The following example shows a CloudTrail Synthetics log entry that demonstrates the `DescribeCanaries` action.


```
{
    "eventVersion": "1.05",
    "userIdentity": {
```

```
        "type": "AssumedRole",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::123456789012:assumed-role/role_name",
        "accountId":"123456789012",
        "accessKeyId":"AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:iam::111222333444:role/Administrator",
                "accountId":"123456789012",
                "userName": "SAMPLE_NAME"
            },
            "webIdFederationData": {},
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2020-04-08T21:43:24Z"
            }
        }
    },
    "eventTime": "2020-04-08T23:06:47Z",
    "eventSource": "synthetics.amazonaws.com",
    "eventName": "DescribeCanaries",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "127.0.0.1",
    "userAgent": "aws-internal/3 aws-sdk-java/1.11.590
 Linux/4.9.184-0.1.ac.235.83.329.metal1.x86_64 OpenJDK_64-Bit_Server_VM/25.212-b03
 java/1.8.0_212 vendor/Oracle_Corporation",
    "requestParameters": null,
    "responseElements": null,
    "requestID": "201ed5f3-15db-4f87-94a4-123456789",
    "eventID": "73ddbd81-3dd0-4ada-b246-123456789",
    "readOnly": true,
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
}
```


The following example shows a CloudTrail Synthetics log entry that demonstrates the `UpdateCanary` action.


```
{
    "eventVersion": "1.05",
    "userIdentity": {
```

```
        "type": "AssumedRole",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::123456789012:assumed-role/role_name",
        "accountId":"123456789012",
        "accessKeyId":"AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
               "type": "Role",
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:iam::111222333444:role/Administrator",
       "accountId":"123456789012",
                "userName": "SAMPLE_NAME"
            },
            "webIdFederationData": {},
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2020-04-08T21:43:24Z"
            }
        }
    },
    "eventTime": "2020-04-08T23:06:47Z",
    "eventSource": "synthetics.amazonaws.com",
    "eventName": "UpdateCanary",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "aws-internal/3 aws-sdk-java/1.11.590
 Linux/4.9.184-0.1.ac.235.83.329.metal1.x86_64 OpenJDK_64-Bit_Server_VM/25.212-b03
 java/1.8.0_212 vendor/Oracle_Corporation",
    "requestParameters": {
        "Schedule": {
            "Expression": "rate(1 minute)"
        },
        "name": "sample_canary_name",
        "Code": {
            "Handler": "myOwnScript.handler",
            "ZipFile": "SAMPLE_ZIP_FILE"
        }
    },
    "responseElements": null,
    "requestID": "fe4759b0-0849-4e0e-be71-1234567890",
    "eventID": "9dc60c83-c3c8-4fa5-bd02-1234567890",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
```

```
}
```

The following example shows a CloudTrail Synthetics log entry that demonstrates the `GetCanaryRuns` action.


```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::123456789012:assumed-role/role_name",
        "accountId":"123456789012",
        "accessKeyId":"AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:iam::111222333444:role/Administrator",
        "accountId":"123456789012",
                "userName": "SAMPLE_NAME"
            },
            "webIdFederationData": {},
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2020-04-08T21:43:24Z"
            }
        }
    },
    "eventTime": "2020-04-08T23:06:30Z",
    "eventSource": "synthetics.amazonaws.com",
    "eventName": "GetCanaryRuns",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "127.0.0.1",
    "userAgent": "aws-internal/3 aws-sdk-java/1.11.590
 Linux/4.9.184-0.1.ac.235.83.329.metal1.x86_64 OpenJDK_64-Bit_Server_VM/25.212-b03
 java/1.8.0_212 vendor/Oracle_Corporation",
    "requestParameters": {
        "Filter": "TIME_RANGE",
        "name": "sample_canary_name",
        "FilterValues": [
            "2020-04-08T23:00:00.000Z",
            "2020-04-08T23:10:00.000Z"
        ]
```

```
    },
    "responseElements": null,
    "requestID": "2f56318c-cfbd-4b60-9d93-1234567890",
    "eventID": "52723fd9-4a54-478c-ac55-1234567890",
    "readOnly": true,
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
}
```


CloudWatch RUM information in CloudTrail

CloudWatch RUM supports logging the following actions as events in CloudTrail log files:

- BatchCreateRumMetricDefinitions
- BatchDeleteRumMetricDefinitions
- BatchGetRumMetricDefinitions
- CreateAppMonitor
- DeleteAppMonitor
- DeleteRumMetricsDestination
- GetAppMonitor
- GetAppMonitorData
- ListAppMonitors
- ListRumMetricsDestinations
- ListTagsForResource
- PutRumMetricsDestination
- TagResource
- UntagResource
- UpdateAppMonitor
- UpdateRumMetricDefinition

Example: CloudWatch RUM log file entries

This section contains example CloudTrail entries for some CloudWatch RUM APIs.

The following example shows a CloudTrail log entry that demonstrates the CreateAppMonitor action.


```
{
    "eventVersion": "1.09",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "EXAMPLE_PRINCIPAL_ID",
        "arn": "arn:aws:sts::777777777777:assumed-role/EXAMPLE",
        "accountId": "777777777777",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "EXAMPLE_PRINCIPAL_ID",
                "arn": "arn:aws:iam::777777777777:role/EXAMPLE",
                "accountId": "777777777777",
                "userName": "USERNAME_EXAMPLE"
            },
            "attributes": {
                "creationDate": "2024-07-23T16:48:47Z",
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2024-07-23T18:02:57Z",
    "eventSource": "rum.amazonaws.com",
    "eventName": "CreateAppMonitor",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "54.240.198.39",
    "userAgent": "aws-internal/3 aws-sdk-java/1.12.641
 Linux/5.10.219-186.866.amzn2int.x86_64 OpenJDK_64-Bit_Server_VM/25.402-b08
 java/1.8.0_402 vendor/Oracle_Corporation cfg/retry-mode/standard",
    "requestParameters": {
        "CustomEvents": {
            "Status": "ENABLED"
        },
        "CwLogEnabled": true,
        "Domain": "*.github.io",
        "AppMonitorConfiguration": {
            "SessionSampleRate": 1,
            "IncludedPages": [],
            "ExcludedPages": [],
```

```
            "Telemetries": [
                "performance",
                "errors",
                "http"
            ],
            "EnableXRay": false,
            "AllowCookies": true,
            "IdentityPoolId": "us-east-1:c81b9a1c-a5c9-4de5-8585-eb8df04e66f0"
        },
        "Tags": {
            "TestAppMonitor": ""
        },
        "Name": "TestAppMonitor"
    },
    "responseElements": {
        "Id": "65a8cc63-4ae8-4f2c-b5fc-4a54ef43af51"
    },
    "requestID": "cf7c30ad-25d3-4274-bab1-39c95a558007",
    "eventID": "2d43cc69-7f89-4f1a-95ae-0fc7e9b9fb3b",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "777777777777",
    "eventCategory": "Management"
}
```


The following example shows a CloudTrail log entry that demonstrates the PutRumMetricsDestination action.


```
{
    "eventVersion": "1.09",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "EXAMPLE_PRINCIPAL_ID",
        "arn": "arn:aws:sts::777777777777:assumed-role/EXAMPLE",
        "accountId": "777777777777",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "EXAMPLE_PRINCIPAL_ID",
                "arn": "arn:aws:iam::777777777777:role/EXAMPLE",
                "accountId": "777777777777",
```

```
                "userName": "USERNAME_EXAMPLE"
            },
            "attributes": {
                "creationDate": "2024-07-23T16:48:47Z",
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2024-07-23T18:22:22Z",
    "eventSource": "rum.amazonaws.com",
    "eventName": "PutRumMetricsDestination",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "52.94.133.142",
    "userAgent": "aws-cli/2.13.25 Python/3.11.5 Linux/5.10.219-186.866.amzn2int.x86_64
 exe/x86_64.amzn.2 prompt/off command/rum.put-rum-metrics-destination",
    "requestParameters": {
        "Destination": "CloudWatch",
        "AppMonitorName": "TestAppMonitor"
    },
    "responseElements": null,
    "requestID": "9b03fcce-b3a2-44fc-b771-900e1702998a",
    "eventID": "6250f9b7-0505-4f96-9668-feb64f82de5b",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "777777777777",
    "eventCategory": "Management"
}
```

The following example shows a CloudTrail log entry that demonstrates the BatchCreateRumMetricsDefinitions action.

```
{
    "eventVersion": "1.09",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "EXAMPLE_PRINCIPAL_ID",
        "arn": "arn:aws:sts::777777777777:assumed-role/EXAMPLE",
        "accountId": "777777777777",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
```

```
                "principalId": "EXAMPLE_PRINCIPAL_ID",
                "arn": "arn:aws:iam::777777777777:role/EXAMPLE",
                "accountId": "777777777777",
                "userName": "USERNAME_EXAMPLE"
            },
            "attributes": {
                "creationDate": "2024-07-23T16:48:47Z",
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2024-07-23T18:23:11Z",
    "eventSource": "rum.amazonaws.com",
    "eventName": "BatchCreateRumMetricDefinitions",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "52.94.133.142",
    "userAgent": "aws-cli/2.13.25 Python/3.11.5 Linux/5.10.219-186.866.amzn2int.x86_64
 exe/x86_64.amzn.2 prompt/off command/rum.batch-create-rum-metric-definitions",
    "requestParameters": {
        "Destination": "CloudWatch",
        "MetricDefinitions": [
            {
                "Name": "NavigationToleratedTransaction",
                "Namespace": "AWS/RUM",
                "DimensionKeys": {
                    "metadata.browserName": "BrowserName"
                },
                "EventPattern": "{\"metadata\":{\"browserName\":[\"Chrome\"]},
\"event_type\":[\"com.amazon.rum.performance_navigation_event\"],\"event_details\":
 {\"duration\": [{\"numeric\": [\"<=\",2000,\"<\",8000]}]}}"
            },
            {
                "Name": "HttpErrorCount",
                "DimensionKeys": {
                    "metadata.browserName": "BrowserName",
                    "metadata.countryCode": "CountryCode"
                },
                "EventPattern": "{\"metadata\":{\"browserName\":[\"Chrome\"],
 \"countryCode\":[\"US\"]},\"event_type\":[\"com.amazon.rum.http_event\"]}"
            }
        ],
        "AppMonitorName": "TestAppMonitor"
    },
    "responseElements": {
```

```
        "Errors": [],
        "MetricDefinitions": []
    },
    "requestID": "b14c5eda-f107-48e5-afae-1ac20d0962a8",
    "eventID": "001b55c6-1de1-48c0-a236-31096dffe249",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "777777777777",
    "eventCategory": "Management"
}
```

CloudWatch RUM data plane events in CloudTrail

CloudTrail can capture API activities related to the CloudWatch RUM data plane operation PutRumEvents.

Data events, also known as data plane operations, give you insight into the resource operations performed on or within a resource. Data events are often high-volume activities.

To enable logging of the **PutRumEvents** data events in CloudTrail files, you'll need to enable the logging of data plane API activity in CloudTrail. See Logging data events for trails for more information.

Data plane events can be filtered by resource type. Because there are additional costs for using data events in CloudTrail, filtering by resource allows you to have more control over what you log and pay for.

Using the information that CloudTrail collects, you can identify a specific request to the CloudWatch RUM **PutRumEvents** API, the IP address of the requester, the requester's identity, and the date and time of the request. Logging the **PutRumEvents** API using CloudTrail helps you enable operational and risk auditing, governance, and compliance of your Amazon account.

The following example shows a CloudTrail log entry that demonstrates the PutRumEvents action.

```
{
 "Records": [
    {
        "eventVersion": "1.09",
        "userIdentity": {
            "type": "AssumedRole",
            "principalId": "EXAMPLE_PRINCIPAL_ID",
```

```
            "arn": "arn:aws:sts::777777777777:assumed-role/EXAMPLE",
            "accountId": "777777777777",
            "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
            "sessionContext": {
                "sessionIssuer": {
                    "type": "Role",
                    "principalId": "EXAMPLE_PRINCIPAL_ID",
                    "arn": "arn:aws:iam::777777777777:role/EXAMPLE",
                    "accountId": "777777777777",
                    "userName": "USERNAME_EXAMPLE"
                },
                "attributes": {
                    "creationDate": "2024-05-16T20:32:39Z",
                    "mfaAuthenticated": "false"
                }
            },
            "invokedBy": "AWS Internal"
        },
        "eventTime": "2024-05-16T20:32:42Z",
        "eventSource": "rum.amazonaws.com",
        "eventName": "PutRumEvents",
        "awsRegion": "us-east-1",
        "sourceIPAddress": "AWS Internal",
        "userAgent": "AWS Internal",
        "requestParameters": {
            "id": "73ddbd81-1234-5678-b246-123456789",
            "batchId": "123456-3dd0-4ada-b246-123456789",
            "appMonitorDetails": {
                "name": "APP-MONITOR-NAME",
                "id": "123456-3dd0-4ada-b246-123456789",
                "version": "1.0.0"
            },
            "userDetails": {
                "userId": "73ddbd81-1111-9999-b246-123456789",
                "sessionId": "a1b2c3456-15db-4f87-6789-123456789"
            },
            "rumEvents": [
                {
                    "id": "201f367a-15db-1234-94a4-123456789",
                    "timestamp": "May 16, 2024, 8:32:20 PM",
                    "type": "com.amazon.rum.dom_event",
                    "metadata": "{}",
                    "details": "{}"
                }
```

```
            ]
        },
        "responseElements": null,
        "requestID": "201ed5f3-15db-4f87-94a4-123456789",
        "eventID": "73ddbd81-3dd0-4ada-b246-123456789",
        "readOnly": false,
        "resources": [
            {
                "accountId": "777777777777",
                "type": "AWS::RUM::AppMonitor",
                "ARN": "arn:aws:rum:us-east-1:777777777777:appmonitor/
APPMONITOR_NAME_EXAMPLE"
            }
        ],
        "eventType": "AwsApiCall",
        "managementEvent": false,
        "recipientAccountId": "777777777777",
        "eventCategory": "Data"
    }
 ]
}
```

Tagging your Amazon CloudWatch resources

A *tag* is a custom attribute label that you or Amazon assigns to an Amazon resource.

Tags help you do the following:

- Identify and organize your Amazon resources. Many Amazon services support tagging. You can assign the same tag to resources from different services to indicate the resources are related. For example, you can assign the tag that you assigned to a CloudWatch rule to an EC2 instance.

Tags have two parts:

- A *tag key* (for example, `CostCenter`, `Environment`, or `Project`). Tag keys are case sensitive.
- An optional field known as a *tag value* (for example, `111122223333` or `Production`). Omitting the tag value is the same as using an empty string. Like tag keys, tag values are case sensitive.

The following sections provide more information about tags for CloudWatch.

Supported resources in CloudWatch

The following resources in CloudWatch support tagging:

- Alarms – You can tag alarms using the tag-resource Amazon CLI command and the TagResource API. You can also view and manage your alarm tags using the *Alarms* details page in the CloudWatch console.
- Canaries – You can tag canaries using the CloudWatch console. For more information, see Creating a canary.
- Contributor Insights rules – You can tag Contributor Insights rules when you create them by using the put-insight-rule Amazon CLI command and the PutInsightRule API. You can add tags to existing rules by using the tag-resource Amazon CLI command and the TagResource API.
- Metric streams – You can tag metric streams when you create them by using the put-metric-stream Amazon CLI command and the PutMetricStream API. You can add tags to existing metric streams by using the tag-resource Amazon CLI command and the TagResource API.

For information about adding and managing tags, see Managing tags.

Managing tags

Tags consist of the `Key` and `Value` properties on a resource. You can use the CloudWatch console, the Amazon CLI, or the CloudWatch API to add, edit, or delete the values for these properties. For information about working with tags, see the following:

- TagResource, UntagResource, and ListTagsForResource in the *Amazon CloudWatch API Reference*
- tag-resource, untag-resource, and list-tags-for-resource in the *Amazon CloudWatch CLI Reference*
- Working with Tag Editor in the *Resource Groups User Guide*

Tag naming and usage conventions

The following basic naming and usage conventions apply to using tags with CloudWatch resources:

- Each resource can have a maximum of 50 tags.
- For each resource, each tag key must be unique, and each tag key can have only one value.
- The maximum tag key length is 128 Unicode characters in UTF-8.
- The maximum tag value length is 256 Unicode characters in UTF-8.
- Allowed characters are letters, numbers, spaces representable in UTF-8, and the following characters: ***. : + = @ _ / -*** (hyphen).
- Tag keys and values are case sensitive. As a best practice, decide on a strategy for capitalizing tags and consistently implement that strategy across all resource types. For example, decide whether to use `Costcenter`, `costcenter`, or `CostCenter` and use the same convention for all tags. Avoid using similar tags with inconsistent case treatment.
- The `aws:` prefix is prohibited for tags because it's reserved for Amazon use. You can't edit or delete tag keys or values with this prefix. Tags with this prefix don't count against your tags per resource limit.

Grafana integration

You can use Grafana version 6.5.0 and later to contextually advance through the CloudWatch console and query a dynamic list of metrics by using wildcards. This can help you monitor metrics for Amazon resources, such as Amazon Elastic Compute Cloud instances or containers. When new instances are created as part of an Auto Scaling event, they appear in the graph automatically. You don't need to track the new instance IDs. Prebuilt dashboards help simplify the getting started experience for monitoring Amazon EC2, Amazon Elastic Block Store, and Amazon Lambda resources.

You can use Grafana version 7.0 and later to perform CloudWatch Logs Insights queries on log groups in CloudWatch Logs. You can visualize your query results in bar, line, and stacked graphs and in a table format. For more information about CloudWatch Logs Insights, see Analyzing Log Data with CloudWatch Logs Insights.

For more information about how to get started, see Using Amazon CloudWatch in Grafana in the Grafana Labs documentation.

CloudWatch service quotas

You can use the table in this section to learn about alarms, alarm email notifications, and API requests in Amazon CloudWatch.

> **Note**
>
> For some Amazon services including CloudWatch, you can use the CloudWatch usage metrics to visualize your current service usage on CloudWatch graphs and dashboards. You can use a CloudWatch metric math function to display the service quotas for those resources on your graphs. You can also configure alarms that alert you when your usage approaches a service quota. For more information, see Visualizing your service quotas and setting alarms.

Resource	Default quota
Alarm actions	5/alarm. This quota can't be changed.
Alarm evaluation period	This is calculated by multiplying the alarm period by the number of evaluation periods used. The maximum value is seven days for alarms with a period of at least one hour (3600 seconds). The maximum is one day for alarms with a shorter period, and the maximum is one day for alarms using the custom Lambda data source. This quota can't be changed.
Alarms	10/month/customer for free. Additional alarms incur charges. No limit on the total number of alarms per account. Alarms based on metric math expressions can have up to 10 metrics. 200 Metrics Insights alarms per Region. You can request a quota increase.

Resource	Default quota
Anomaly detection models	500 per Region, per account.
API requests	1,000,000/month/customer for free.
Application Signals API requests	The following APIs have a quota of 10 transactions per second (TPS) and per Region. You can request a quota increase. • BatchGetServiceLevelObjectiveBudgetReport • GetService • ListServiceDependencies • ListServiceDependents • ListServiceOperations • ListServices The following APIs have a quota of 5 transactions per second (TPS) and per Region. You can request a quota increase. • CreateServiceLevelObjective • DeleteServiceLevelObjective • GetServiceLevelObjective • ListServiceLevelObjectives • ListTagsForResource • StartDiscovery • TagResource • UntagResource • UpdateServiceLevelObjective

Resource	Default quota
SLOs	Maximum number of SLOs supported per Region per account: **250** Maximum number of SLOs supported per Service: **100** You can request a quota increase.
Canaries	200 per Region per account. You can request a quota increase.
Contributor Insights API requests	The following APIs have a quota of 20 transactions per second (TPS) and per Region. • DescribeInsightRules The quota can't be changed. • GetInsightRuleReport You can request a quota increase. The following APIs have a quota of 5 TPS per Region. This quota can't be changed. • DeleteInsightRules • PutInsightRule The following APIs have a quota of 1 TPS per Region. This quota can't be changed. • DisableInsightRules • EnableInsightRules

Resource	Default quota
Contributor Insights rules	100 rules per Region per account. You can request a quota increase.
Custom metrics	No quota.
Dashboards	Up to 500 widgets per dashboard. Up to 500 metrics per dashboard widget. Up to 2500 metrics per dashboard, across all widgets. These quotas include all metrics retrieved for use in metric math functions, even if those metrics are not displayed on the graph. These quotas can't be changed.
DescribeAlarms	9 transactions per second (TPS) per Region. The maximum number of operation requests you can make per second without being throttled. You can request a quota increase.
DescribeAlarmHistory	20 transactions per second (TPS) per Region. The maximum number of operation requests you can make per second without being throttled. This quota can't be changed.
DeleteAlarms request DisableAlarmActions request EnableAlarmActions request SetAlarmState request	3 TPS per Region for each of these operations. The maximum number of operation requests you can make per second without being throttled. These quotas can't be changed.

Resource	Default quota
DescribeAlarmsForMetric request	9 TPS per Region. The maximum number of operation requests you can make per second without being throttled. This quota can't be changed.
DeleteDashboards request GetDashboard request ListDashboards request PutDashboard request	10 TPS per Region for each of these operations. The maximum number of operation requests you can make per second without being throttled. These quotas can't be changed.
PutAnomalyDetector DescribeAnomalyDetectors	10 TPS per Region. The maximum number of operation requests you can make per second without being throttled.
DeleteAnomalyDetector	5 TPS per Region. The maximum number of operation requests you can make per second without being throttled.
Dimensions	30/metric. This quota can't be changed.

Resource	Default quota
GetMetricData	10 TPS per Region for operations that include Metrics Insights queries. For operations that do not include Metrics Insights queries, the quota is 50 TPS per Region. This is the maximum number of operation requests you can make per second without being throttled. You can request a quota increase. For `GetMetricData` operations that include a Metrics Insights query, the quota is 4,300,000 Datapoints Per Second (DPS) for the most recent 3 hours. This is calculated against the total number of data points scanned by the query (which can include no more than 10,000 metrics.) 180,000 Datapoints Per Second (DPS) if the `StartTime` used in the API request is less than or equal to three hours from current time. 396,000 DPS if the `StartTime` is more than three hours from current time. This is the maximum number of datapoints you can request per second using one or more API calls without being throttled. This quota can't be changed. The DPS is calculated based on estimated data points, not actual data points. The data point estimate is calculated using the requested time range, period, and retention period. This means that if the actual data points in the requested metrics are sparse or empty, throttling still occurs if the estimated data points exceed the quota. The DPS quota is per-Region.

Resource	Default quota
GetMetricData	A single `GetMetricData` call can include the following: • As many as 500 `MetricDataQuery` structures. • As many as 100 `SERVICE_QUOTA()` functions. • As many as 100 `SEARCH()` functions. • As many as 5 `LAMBDA()` functions. These quotas can't be changed.
GetMetricStatistics	400 TPS per Region. The maximum number of operation requests you can make per second without being throttled. You can request a quota increase.
GetMetricWidgetImage	Up to 500 metrics per image. This quota can't be changed. 20 TPS per Region. The maximum number of operation requests you can make per second without being throttled. You can request a quota increase.
ListMetrics	25 TPS per Region. The maximum number of operation requests you can make per second without being throttled. You can request a quota increase.
Metric data values	The value of a metric data point must be within the range of -2^360 to 2^360. Special values (for example, NaN, +Infinity, -Infinity) are not supported. This quota can't be changed.

Resource	Default quota
MetricDatum items	1000/PutMetricData request. A MetricDatum object can contain a single value or a StatisticSet object represent ing many values. This quota can't be changed.
Metrics	10/month/customer for free.
Metrics Insights queries	A single query can process no more than 10,000 metrics. This means that if the **SELECT**, **FROM**, and **WHERE** clauses would match more than 10,000 metrics, only the first 10,000 of these metrics that are found will be processed by the query. A single query can return no more than 500 time series. You can query only the most recent three hours of data
Metrics Insights query generator requests	As many as five concurrent natural-language generated requests.
Observability Access Manager (OAM) API request rates.	1 TPS per Region for PutSinkPolicy. 10 TPS per Region for each other CloudWatch OAM API. These quotas reflect the maximum number of operation requests you can make per second without being throttled. These quotas can't be changed.
OAM source account links	Each source account can be linked to as many as 5 monitoring accounts This quota can't be changed.
OAM sinks	1 sink per Region per account This quota can't be changed.

Resource	Default quota
PutCompositeAlarm request	3 TPS per Region. The maximum number of operation requests you can make per second without being throttled. You can request a quota increase.
PutPutDashboard payload size	The payload of `PutDashboard` requests can be up to 1 MB. This quota can't be changed.
PutMetricAlarm request	3 TPS per Region. The maximum number of operation requests you can make per second without being throttled. You can request a quota increase.
PutMetricData request	1MB for HTTP POST requests. PutMetricData can handle 500 transactions per second (TPS), which is the maximum number of operation requests that you can make per second without being throttled. `PutMetricData` can handle 1,000 metrics per request. You can request a quota increase.
Amazon Q Developer operational investigations investigations	You can have as many as 20 active investigations. Only two of these investigations at a time can have active analysis currently happening.
Amazon SNS email notifications	1,000/month/customer for free.
Synthetics Groups	20 per account. This quota can't be changed.

Resource	Default quota
TagResource	20 TPS per Region. The maximum number of operation requests you can make per second without being throttled. This quota can't be changed.
UntagResource	20 TPS per Region. The maximum number of operation requests you can make per second without being throttled. This quota can't be changed.

Document history

The following table describes important changes for each release of the *Amazon CloudWatch User Guide*, beginning in June 2018. To receive notifications about updates to this documentation, you can subscribe to an RSS feed.

Change	Description	Date
CloudWatch Network Monitoring releases multiple account support for Network Flow Monitor	Network Flow Monitor, a feature of CloudWatch Network Monitoring, now supports monitoring network performance metrics for network flows for resources in multiple accounts. For more information, see Initialize Network Flow Monitor for multi-account monitoring.	May 6, 2025
AmazonCloudWatchRUMReadOnlyAccess updated	The `rum:GetResourcePolicy` permission was added so that CloudWatch RUM can view the resource policy attached to the RUM application monitor.	April 28, 2025
CloudWatch Synthetics deprecates five runtime metrics on October 1, 2025	CloudWatch Synthetics metrics `syn-nodejs-puppeteer-7.0` , `syn-nodejs-puppeteer-6.2` , `syn-nodejs-puppeteer-5.2` , `syn-python-selenium-3.0` , and `syn-python-selenium-2.1` will be deprecated on October 1, 2025.	April 4, 2025

Updated CloudWatchApplicationSignalsServiceRolePolicy policy	CloudWatch updated the policy to exclude time windows from impacting the SLO attainment rate, error budget, and burn rate metrics. CloudWatch can retrieve exclusion windows on behalf of you. For more information, see CloudWatchApplicationSignalsServiceRolePolicy.	March 13, 2025
Time window exclusions	SLOs adds support for time window exclusions which is a block of time with a defined start and end date. This time period is excluded from the SLO's performance metrics and you can schedule one-time or recurring time exclusions windows. For more information, see Service level objectives (SLOs).	March 13, 2025
New AIOpsReadOnlyAccess policy	CloudWatch added the **AIOpsReadOnlyAccess** policy, which grants read-only permissions for Amazon AI Operations and other related services.	December 3, 2024

New AIOpsOperatorAccess policy	CloudWatch added the **AIOpsOperatorAccess** policy, which grants users access to Amazon AI Operations actions, and to additional Amazon actions that are necessary for accessing investigation events.	December 3, 2024
New AIOpsConsoleAdminPolicy policy	CloudWatch added the **AIOpsConsoleAdminPolicy** policy, which grants users full administrative access for managing Amazon Q investigations, including the management of trusted identity propagation, and the management of IAM Identity Center and organizational access.	December 3, 2024
New AIOpsAssistantPolicy policy	CloudWatch added the **AIOpsAssistantBasicAccess** policy, which you can assign to the Amazon AI Operations assistant to enable the assistant to have basic functionality.	December 3, 2024

Amazon Q Developer operational investigations feature in Preview	CloudWatch released a Preview of Amazon Q Developer operational investigations. This feature is a generative AI-powered assistant that can help you respond to incidents in your system. It uses generativ e AI to scan your system's telemetry and quickly surface suggestions that might be related to your issue, to help you troubleshoot operational issues faster.	December 3, 2024
CloudWatchFullAccessV2 and CloudWatchFullAccess policies updated	CloudWatch updated both **CloudWatchFullAccessV2** and **CloudWatchFullAcce ss**. Permissions for Amazon OpenSearch Service were added to to enable CloudWatch Logs integration with OpenSearch Service for some features.	December 1, 2024
Telemetry config feature generally available	CloudWatch added the telemetry config feature to discover and understan d the state of telemetry configuration for your Amazon resources.	November 26, 2024
New AWSServiceRoleForO bservabilityAdmin service-l inked role	CloudWatch added the new AWSServiceRoleForObservabil ityAdmin service-linked role and AWSObservabilityAd minServiceRolePolicy policy.	November 26, 2024

CloudWatch supports exploring related telemetry	CloudWatch supports exploring related telemetry data across interconnected resources.	November 22, 2024
Updated CloudWatchSyntheticsFullAccess IAM policy	CloudWatch updated the **CloudWatchSyntheticsFullAccess** policy to grant CloudWatch Synthetics access to get Amazon Lambda functions, and perform actions on logs available in CloudWatch Logs. Additionally, Lambda layer version actions apply to all CloudWatch Synthetics layer ARNs.	November 21, 2024
New CloudWatch Synthetics support for Node.js with the Playwright runtime	CloudWatch Synthetics adds support for Node.js canary scripts that use the Playwright runtime.	November 21, 2024
CloudWatchReadOnlyAccess policy updated	The scope of the policy was changed to include the `xray:List *`, `xray:StartTraceRetrieval` , and `xray:CancelTraceRetrieval` permissions. This change allows CloudWatch to access new X-Ray read APIs. For more infomration, see CloudWatchReadOnlyAccess.	November 21, 2024

CloudWatch Transaction Search is generally available	CloudWatch releases Transacti on Search, a search and analytics experience. Transacti on Search provides visibilit y into application transacti on issues. Explore interacti ons between application components to establish a root cause. Understand your application transactions and their impact on customers . Search transactions using terms like customer name or order. For more information, see CloudWatch Transaction Search.	November 21, 2024
Application Signals supports Lambda functions	You can now enable Applicati on Signals for Lambda functions.	November 21, 2024
OpenTelemetry	You can use OpenTelemetry to directly send traces to an OpenTelemetry Protocol (OTLP) endpoint, and get out-of-the box application performance monitoring experiences in CloudWatch Application Signals.	November 20, 2024
Application Signals collects runtime metrics	Application Signals now collects OpenTelemetry-compatible runtime metrics from your Java and Python applications.	November 19, 2024

Application Signals collects runtime metrics	Application Signals now collects OpenTelemetry-compatible runtime metrics from your Java and Python applications.	November 19, 2024
Internet traffic monitoring for VPCs in Amazon Local Zones	Internet Monitor now supports internet traffic monitoring for VPCs deployed in Amazon Local Zones	November 18, 2024
Application Signals support for Node.js applications is generally available	CloudWatch supports enabling Node.js applications for Application Signals.	November 15, 2024
New CloudWatchInternet MonitorReadOnlyAccess IAM policy	CloudWatch added a **CloudWatchInternet MonitorReadOnlyAcc ess** policy, to grant read-only access to actions and resources available in the CloudWatch console for Internet Monitor.	November 14, 2024
CloudWatch added observabi lity solutions	CloudWatch observability solutions offer a catalog of readily available configura tions to help you quickly implement monitoring for various Amazon services and common workloads, such as Java Virtual Machines (JVM), Apache Kafka, Apache Tomcat, and NGINX.	November 13, 2024

CloudWatch added burn rate monitoring for Application Signals SLOs	CloudWatch added burn rate monitoring to Application Signals SLOs. A burn rate is a metric that indicates how fast the service is consuming the error budget, relative to the attainment goal of the SLO.	November 13, 2024
New Internet Monitor latency reduction suggestions through Route 53 routing updates	Amazon CloudWatch Internet Monitor now provides actionable suggestions to help you optimize your Route 53 IP-based routing configurations. Easily identify the optimal Amazon Web Services Regions for routing your end user traffic, and then configure your Route 53 IP-based routing based on the recommendations.	November 1, 2024
New CloudWatchInternet MonitorFullAccess IAM policy	CloudWatch added a **CloudWatchInternet MonitorFullAccess** policy, to grant full access to actions and resources available in the CloudWatch console for Internet Monitor.	October 23, 2024
End of support notice	End of support notice: On October 16, 2025, Amazon will discontinue support for CloudWatch Evidently. After October 16, 2025, you will no longer be able to access the Evidently console or Evidently resources.	October 17, 2024

End of support notice	End of support notice: On October 16, 2025, Amazon will discontinue support for CloudWatch Evidently. After October 16, 2025, you will no longer be able to access the Evidently console or Evidently resources.	October 17, 2024
New CloudWatchLambdaApplicationSignalsExecutionRolePolicy IAM policy	CloudWatch added a new policy named **CloudWatchLambdaApplicationSignalsExecutionRolePolicy**. This policy is used when CloudWatch Application Signals is enabled for Lambda workloads. It enables write access to X-Ray and the log group used by CloudWatch Application Signals.	October 16, 2024
New CloudWatchLambdaApplicationSignalsExecutionRolePolicy IAM policy	CloudWatch added a new policy named **CloudWatchLambdaApplicationSignalsExecutionRolePolicy**. This policy is used when CloudWatch Application Signals is enabled for Lambda workloads. It enables write access to X-Ray and the log group used by CloudWatch Application Signals.	October 16, 2024

CloudWatchSyntheticsFullAccess policy updated	Permissions were added to the **CloudWatchSyntheticsFullAccess** IAM policy. The `lambda:ListTags` , `lambda:TagResource` , and `lambda:UntagResource` permissions were added so that when you apply or change tags on a canary, you can choose to have Synthetics also apply those same tags or changes to the Lambda function that the canary uses.	October 11, 2024
Application Signals supports request-based SLOs	Application Signals adds support for request-based service level objects (SLOs). For more information, see Service level objectives (SLOs).	September 6, 2024
Internet Monitor refreshed dashboard and latency improvement suggestions	Amazon CloudWatch Internet Monitor has launched an updated console experience, including new features for visualizing configuration changes that can help you reduce latency for your application.	August 12, 2024
Application Signals previewing support for .NET applications	CloudWatch Application Signals has added support for .NET applications, with ADOT Instrumentation for .NET on supported platforms. This feature is in preview release.	August 5, 2024

New CloudWatch Applicati on Insights service linked role permission added	Allows CloudWatch Applicati on Insights to enable and disable termination protectio n on Amazon CloudFormation stacks.	July 25, 2024
CloudWatch Metrics Insights support for natural language query generation is generally available.	CloudWatch Metrics Insights supports natural language query to generate and update queries. For more informati on, see Use natural language to generate and update CloudWatch Metric Insights queries.	June 10, 2024
CloudWatch Application Signals is generally available	CloudWatch Application Signals is now generally available. Use Application Signals to instrument your applications on Amazon so that you can monitor current application health, create service level objectives (SLOs), and track long-term applicati on performance against your business objectives. For more information, see Application Signals.	June 10, 2024
New CloudWatchApplicat ionSignalsReadOnlyAccess IAM policy	CloudWatch added a **CloudWatchApplicat ionSignalsReadOnlyAccess** policy to add read only actions and resources available in the CloudWatc h console for Application Signals.	June 7, 2024

New CloudWatchApplicationSignalsFullAccess policy	CloudWatch added a **CloudWatchApplicationSignalsFullAccess** policy to add actions and resources available in the CloudWatch console for Application Signals.	June 7, 2024
CloudTrail now captures API activities related to CloudWatch data plane operations.	CloudTrail now logs events in CloudTrail for **GetMetricData** and **GetMetricWidgetImage** API activities.	June 6, 2024
CloudWatch Application Signals service map supports canary, RUM clients and Amazon service dependency groupings.	The Application Signals preview release has added default groupings in the service map for canaries, RUM clients, and Amazon service dependencies of the same type. This change reduces the number of icons in the service map default view to make it easier to view and navigate.	May 21, 2024

CloudWatchReadOnlyAccess policy updated	CloudWatch changed the scope of a permission in **CloudWatchReadOnlyAccess**. The scope of the policy added the `application-signals:BatchGet*` , `application-signals:Get*` , and `application-signals:List*` actions so that users can use CloudWatch Application Signals to view, investigate, and diagnose issues with the health of their services. CloudWatch also added an `iam:GetRole` action so that users can check if Application Signals is set up.	May 17, 2024
CloudWatchFullAccessV2 policy updated	CloudWatch changed the scope of a permission in **CloudWatchFullAccessV2**. The scope of the policy added the `application-signals:*` so that users can use CloudWatch Application Signals to view, investigate, and diagnose issues with the health of their services.	May 17, 2024
Lambda Insights supports Amazon GovCloud (US-East) and Amazon GovCloud (US-West)	CloudWatch Lambda Insights has added support for the Amazon GovCloud (US-East) and Amazon GovCloud (US-West) Regions.	April 29, 2024

CloudWatch cross-account observability supports resource filters	You can now create filters to specify which metric namespaces and log groups are shared from the source account to the monitoring account, when you create the link between the accounts.	April 26, 2024
CloudWatch Application Signals updates	The Application Signals preview release has added three features. Applicati on Signals now supports Python applications. It offers a simpler enablement process for applications on Amazon EKS architectures. And it includes new configurations that you can use to manage the cardinality of metrics that are collected.	April 26, 2024
CloudWatch Container Insights with enhanced observability for Amazon EKS can collect Amazon Elastic Fabric Adapter (EFA) metrics	You can now use CloudWatc h Container Insights with enhanced observability for Amazon EKS collect Amazon Elastic Fabric Adapter (EFA) metrics from Amazon EKS clusters.	April 23, 2024

Updated IAM policy	CloudWatch updated the **CloudWatchApplicationSignalsServiceRolePolicy** policy. The scoping of the `logs:StartQuery` and `logs:GetQueryResults` permissions in this policy was changed to add `arn:aws:logs:*:*:log-group:/aws/appsignals/*:*` and `"arn:aws:logs:*:*:log-group:/aws/application-signals/data:*"` to enable Application Signals on more architecures. This policy is attached to the **AWSServiceRoleForCloudWatchApplicationSignals** service-linked role.	April 18, 2024
Internet Monitor provides a global internet weather map to authenticated Amazon customers	Amazon CloudWatch Internet Monitor now displays a global internet weather map that is available in the console to all authenticated Amazon customers. To view the map, in the Amazon CloudWatch console, navigate to Internet Monitor.	April 16, 2024

CloudWatch Container Insights with enhanced observability for Amazon EKS can collect Amazon Neuron metrics	You can now use CloudWatch Container Insights with enhanced observability for Amazon EKS collect Amazon Neuron metrics from Amazon EKS clusters.	April 16, 2024
CloudWatch Application Signals adds a **Service overview** tab and more metrics to aid in diagnostics	A new **Service overview** tab displays an overview of your service, including number of operations, dependencies, synthetics, and client pages. The tab shows key metrics for your entire service, and top operations and dependencies. You can also now view X-Ray traces that are correlated with issues including faults, errors, and latency problems.	April 16, 2024
CloudWatch Container Insights with enhanced observability for Amazon EKS adds support for Windows	You can now use CloudWatch Container Insights with enhanced observability for Amazon EKS collect metrics from Windows worker nodes on Amazon EKS clusters.	April 10, 2024
CloudWatchApplicationSignalsServiceRolePolicy IAM policy updated	CloudWatch changed the scope of a permission in **CloudWatchApplicationSignalsServiceRolePolicy**. The scope of the `cloudwatch:GetMetricData` permission was changed to * so that Application Signals can retrieve metrics from sources in linked accounts.	April 8, 2024

Amazon CloudWatch Internet Monitor now supports cross-account observability	You can now use Internet Monitor cross-account observability to monitor your applications that span multiple Amazon Web Services accounts within a single Amazon Web Services Region.	March 29, 2024
New service linked role and IAM policy for CloudWatch Network Monitor	CloudWatch added a new service-linked role, called **AWSServiceRoleForNetworkMonitor**. CloudWatch added this new service-linked role to allow you to create monitors to fetch network metrics between source subnets and destination IP addresses. The new **CloudWatchNetworkMonitorServiceRolePolicy** IAM policy is attached to this role, and the policy grants permission to CloudWatch to fetch network metrics on your behalf.	December 22, 2023
CloudWatch releases Amazon CloudWatch Network Monitor	CloudWatch released a new feature, Amazon CloudWatch Network Monitor. This is a new active network monitoring service that identifies if a network issues exists within the Amazon network or your own company network.	December 22, 2023

CloudWatchReadOnlyAccess policy updated	CloudWatch added existing read-only permissions for CloudWatch Synthetics, X-Ray, and CloudWatch RUM and new read-only permissions for CloudWatch Application Signals to**CloudWatchReadOnlyAccess** so that users with this policy can triage and diagnose service health issues as reported by CloudWatch Application Signals. The `cloudwatch:GenerateQuery` permission was added so that users with this policy can generate a CloudWatch Metrics Insights query string from a natural language prompt.	December 5, 2023
CloudWatchFullAccessV2 policy updated	CloudWatch added existing permissions to **CloudWatchFullAccessV2** for CloudWatch Synthetics, X-Ray, and CloudWatch RUM, and added new permissions for CloudWatch Application Signals so that users with this policy can fully manage Application Signals to triage and diagnose issues with service health.	December 5, 2023

New service-linked role and new IAM policy	CloudWatch added a new service-linked role, called **AWSServiceRoleForCloudWatchApplicationSignals**. CloudWatch added this new service-linked role to allow CloudWatch Application Signals to collect CloudWatch Logs data, X-Ray trace data, CloudWatch metrics data, and tagging data from applications that you have enabled for CloudWatch Application Signals. The new **CloudWatchApplicationSignalsServiceRolePolicy** IAM policy is attached to this role, and the policy grants permission to CloudWatch Application Signals to collect monitoring and tagging data from other relevant Amazon services.	November 30, 2023

CloudWatch launches Preview release of Application Signals	CloudWatch Application Signals is in Preview. Use Application Signals to instrument your applications on Amazon so that you can monitor current application health, create service level objectives (SLOs), and track long-term application performance against your business objectives. For more information, see Application Signals.	November 30, 2023
CloudWatch adds support for querying other data sources	You can use CloudWatch to query, visualize, and create alarms for metrics from other data sources For more information, see Querying metrics from other data sources.	November 26, 2023
CloudWatch Metrics Insights supports natural language query generation	CloudWatch Metrics Insights supports natural language query to generate and update queries. For more information, see Use natural language to generate and update CloudWatch Metric Insights queries.	November 26, 2023

CloudWatch releases Container Insights with enhanced observability for Amazon EKS	CloudWatch released a new version of Container Insights. This version supports enhanced observability for Amazon EKS clusters and can collect more detailed metrics from clusters running Amazon EKS. After installat ion, it automatically collects detailed infrastructure telemetry and container logs for your Amazon EKS clusters. You can then use curated, immediately usable dashboards to drill down into application and infrastructure telemetry.	November 6, 2023
CloudWatch metric streams adds quick partner setup	CloudWatch metric streams now provides a quick partner setup option, which you can use to quickly set up a metric stream to some third-party providers.	October 17, 2023
CloudWatch releases alarm recommendations	CloudWatch Synthetics now provides alarm recommend ations for metrics from other Amazon services. These recommendations can help you identify the metrics that you should set alarms for to follow best practices for monitoring these services.	October 16, 2023

CloudWatch Synthetics releases runtime syn-nodejs-puppeteer-6.0	CloudWatch Synthetics released runtime `syn-nodejs-puppeteer-6.0` .	September 26, 2023
Adds Amazon CloudWatch Application Insights support for cross-account applications	You can now share CloudWatch Application Insights applications across account boundaries.	September 26, 2023
New service-linked role and new IAM policy	CloudWatch added a new service-linked role, called **AWSServiceRoleForCloudWatchMetrics_DbPerfInsights**. CloudWatch added this new service-linked role to allow CloudWatch to fetch Performance Insights metrics for alarming, anomaly detection, and snapshotting. The new **AWSServiceRoleForCloudWatchMetrics_DbPerfInsightsServiceRolePolicy** IAM policy is attached to this role, and the policy grants permission to CloudWatch to fetch Performance Insights metrics on your behalf.	September 20, 2023

Adds new metric math function	CloudWatch added a new metric math function, `DB_PERF_INSIGHTS` , that you can use to fetch Performance Insights metrics from Amazon database services for alarming, anomaly detection, and snapshotting.	September 20, 2023
CloudWatchReadOnlyAccess policy updated	CloudWatch added the `application-autoscaling:DescribeScalingPolicies` permission to **CloudWatchReadOnlyAccess** so that users with this policy can access information about Application Auto Scaling policies.	September 14, 2023
CloudWatch agent added support for AL2023	The CloudWatch agent supports AL2023.	August 8, 2023
New managed IAM policy, CloudWatchFullAccessV2	CloudWatch added a new policy **CloudWatchFullAccessV2**. This policy grants full access to CloudWatch actions and resources while better scoping the permissions granted to other services such as Amazon SNS and Amazon EC2 Auto Scaling.	August 1, 2023

Updated service linked role for Amazon CloudWatch Internet Monitor – update to an existing policy	Adds new permissions, `elasticloadbalancing:DescribeLoadBalancers` and `ec2:DescribeNetworkInterfaces`, to the service linked role for Internet Monitor, to support monitoring traffic for specific Network Load Balancer resources.	July 25, 2023
Added support for Network Load Balancer resources in Amazon CloudWatch Internet Monitor	Adds support for creating a monitor in Internet Monitor with specific Network Load Balancer resources, to provide more granular levels of observability for your application.	July 25, 2023
Dashboard variables feature	CloudWatch released *dashboard variables*, which you can use to create flexible dashboards that can quickly display different contents depending on how you set one input field within the dashboard. For example, you can create a dashboard that can quickly switch between different Lambda functions or Amazon EC2 instance IDs, or one that can switch to different Amazon Regions. For more information, see Create flexible dashboards with dashboard variables.	June 28, 2023

Internet Monitor now supports customizing the threshold for health events	Internet Monitor added the ability to customize the threshold for when a global performance score or availability score triggers a heath event. For more information, see Tracking real-time performance and availability in Amazon CloudWatch Internet Monitor.	June 26, 2023
Internet Monitor now supports all commercial Regions	Internet Monitor added seven new Amazon Web Services Regions and now supports all commercial Regions.	June 19, 2023
New Lambda Insights extension versions	CloudWatch added the 1.0.229.0 version of the Lambda Insights extension for both x86-64 platforms and ARM64 platforms. For more information, see Available versions of the Lambda Insights extension.	June 12, 2023
CloudWatchReadOnlyAccess policy updated	CloudWatch added permissions to **CloudWatchReadOnlyAccess**. The `logs:StartLiveTail` and `logs:StopLiveTail` permissions were added so that users with this policy can use the console to start and stop CloudWatch Logs live tail sessions. For more information, see Use live tail to view logs in near real time.	June 6, 2023

CloudWatch RUM adds support for custom metrics	You can use CloudWatch RUM app monitors to create custom metrics and send them to CloudWatch and CloudWatch Evidently. This feature includes an update to the **AmazonCloudWatchRUMServiceRolePolicy** managed IAM policy. In that policy, a condition key was changed so that CloudWatch RUM can send custom metrics to custom metric namespaces.	February 9, 2023
New and updated managed polices for CloudWatch	To support CloudWatch cross-account observability, the `CloudWatchFullAccess` and `CloudWatchReadOnlyAccess` policies have been updated, and the following new managed policies have been added: `CloudWatchCrossAccountSharingConfiguration` , `OAMFullAccess` , and `OAMReadOnlyAccess` . For more information, see CloudWatch updates to Amazon managed policies.	February 7, 2023
CloudWatch Application Insights service-linked role policy updates — update to an existing policy.	CloudWatch Application Insights updated an existing Amazon service-linked role policy.	December 19, 2022

Amazon CloudWatch Application Insights support for containerized applications and microservices from the Container Insights console.	You can display CloudWatch Application Insights detected problems for Amazon ECS and Amazon EKS on your Container Insights dashboard.	November 17, 2021
Amazon CloudWatch Application Insights monitoring for SAP HANA databases.	You can monitor SAP HANA databases with Application Insights.	November 15, 2021
Amazon CloudWatch Application Insights support for monitoring all resources in an account.	You can onboard and monitor all resources in an account.	September 15, 2021
Amazon CloudWatch Application Insights support for Amazon FSx.	You can monitor metrics retrieved from Amazon FSx.	August 31, 2021
SDK Metrics is no longer supported.	CloudWatch SDK Metrics is no longer supported.	August 25, 2021
Amazon CloudWatch Application Insights support for setting up container monitoring.	You can monitor containers using best practices with Amazon CloudWatch Application Insights.	May 18, 2021
Metric streams is generally available	You can use metric streams to continually stream CloudWatch metrics to a destination of your choice. For more information, see Metric streams in the *Amazon CloudWatch User Guide*.	March 31, 2021

Amazon CloudWatch Application Insights monitoring for Oracle databases on Amazon RDS and Amazon EC2.	You can monitor metrics and logs retrieved from Oracle with Amazon CloudWatch Application Insights.	January 16, 2021
Lambda Insights is generally available	CloudWatch Lambda Insights is a monitoring and troubleshooting solution for serverless applications running on Amazon Lambda. For more information, see Using Lambda Insights in the *Amazon CloudWatch User Guide*.	December 3, 2020
Amazon CloudWatch Application Insights monitoring for Prometheus JMX exporter metrics.	You can monitor metrics retrieved from Prometheus JMX exporter with Amazon CloudWatch Application Insights.	November 20, 2020
CloudWatch Synthetics releases new runtime version	CloudWatch Synthetics has released a new runtime version. For more information, see Canary Runtime Versions in the *Amazon CloudWatch User Guide*.	September 11, 2020
Amazon CloudWatch Application Insights monitoring for Postgre SQL on Amazon RDS and Amazon EC2.	You can monitor applications built with PostgreSQL running on Amazon RDS or Amazon EC2.	September 11, 2020

CloudWatch supports dashboard sharing	You can now share CloudWatch dashboards with people outside of your organization and Amazon account. For more information, see Sharing CloudWatch Dashboards in the *Amazon CloudWatch User Guide*.	September 10, 2020
Set up monitors for .NET applications using SQL Server on the backend with CloudWatch Application Insights	You can use the documentation tutorial to help you to set up monitors for .NET applications using SQL Server on the backend with CloudWatch Application Insights.	August 19, 2020
Amazon CloudFormation support for Amazon CloudWatch Application Insights applications.	You can add CloudWatch Application Insights monitoring, including key metrics and telemetry, to your application, database, and web server, directly from Amazon CloudFormation templates.	July 30, 2020
Amazon CloudWatch Application Insights monitoring for Aurora for MySQL database clusters.	You can monitor Aurora for MySQL database clusters (RDS Aurora) with Amazon CloudWatch Application Insights.	July 2, 2020

CloudWatch Contributor Insights general availability	CloudWatch Contributor Insights is now generally available. It enables you to analyze log data and create time series that display contributor data. You can see metrics about the top-N contributors, the total number of unique contributors, and their usage. For more information, see Using Contributor Insights to Analyze High-Cardinality Data in the *Amazon CloudWatch User Guide*.	April 2, 2020
CloudWatch Synthetics public preview	CloudWatch Synthetics is now in public preview. It enables you to create canaries to monitor your endpoints and APIs. For more information, see Using Canaries in the *Amazon CloudWatch User Guide*.	November 25, 2019

CloudWatch Contributor Insights public preview	CloudWatch Contributor Insights is now in public preview. It enables you to analyze log data and create time series that display contributor data. You can see metrics about the top-N contributors, the total number of unique contributors, and their usage. For more information, see Using Contributor Insights to Analyze High-Cardinality Data in the *Amazon CloudWatch User Guide*.	November 25, 2019
CloudWatch launches ServiceLens feature	ServiceLens ehances the observability of your services and applications by enabling you to integrate traces, metrics, logs, and alarms into one place. ServiceLens integrates CloudWatch with Amazon X-Ray to provide an end-to-end view of your application.	November 21, 2019

Use CloudWatch to proactively manage your Amazon service quotas	You can use CloudWatch to proactively manage your Amazon service quotas. CloudWatch usage metrics provide visibility into your account's usage of resources and API operations. For more information, see Service Quotas Integration and Usage Metrics in the *Amazon CloudWatch User Guide*.	November 19, 2019
CloudWatch sends events when alarms change state	CloudWatch now sends an event to Amazon EventBridge when any CloudWatch alarm changes state. For more information, see Alarm Events and EventBridge in the *Amazon CloudWatch User Guide*.	October 8, 2019
Container Insights	CloudWatch Container Insights is now generally available. It enables you to collect, aggregate, and summarize metrics and logs from your containerized applications and microservices. For more information, see Using Container Insights in the *Amazon CloudWatch User Guide*.	August 30, 2019

Updates for Container Insights preview metrics on Amazon EKS and Kubernetes	The Container Insights on Amazon EKS and Kubernetes public preview has been updated. InstanceId is now included as a dimension to the cluster EC2 instances . This allows alarms that have been created on these metrics to trigger the following EC2 actions: Stop, Terminate, Reboot, or Recover. Additionally, pod and service metrics are now reported by Kubernetes namespace to simplify the monitoring and alarming on metrics by namespace.	August 19, 2019
Updates for Amazon Systems Manager OpsCenter integration	Updates on how CloudWatch Application Insights integrates with Systems Manager OpsCenter.	August 7, 2019
CloudWatch usage metrics	CloudWatch usage metrics help you track the usage of your CloudWatch resources and stay within your service limits. For more information, see https://docs.amazonaws.cn/AmazonCloudWatch/latest/monitoring/CloudWatch-Usage-Metrics.html.	August 6, 2019

CloudWatch Container Insights public preview	CloudWatch Container Insights is now in public preview. It enables you to collect, aggregate, and summarize metrics and logs from your containerized applications and microserv ices. For more information, see Using Container Insights in the *Amazon CloudWatch User Guide*.	July 9, 2019
CloudWatch Anomaly Detection public preview	CloudWatch anomaly detection is now in public preview. CloudWatch applies machine-learning algorithm s to a metric's past data to create a model of the metric's expected values. You can use this model for visualiza tion and for setting alarms. For more information, see Using CloudWatch Anomaly Detection in the *Amazon CloudWatch User Guide*.	July 9, 2019
CloudWatch Application Insights for .NET and SQL Server	CloudWatch Application Insights for .NET and SQL Server facilitates observabi lity for .NET and SQL Server applications. It can help you set up the best monitors for your application resources to continuously analyze data for signs of problems with your applications.	June 21, 2019

CloudWatch agent section re-organized	The CloudWatch agent documentation has been rewritten to improve clarity, especially for customers using the command line to install and configure the agent. For more information, see Collecting Metrics and Logs from Amazon EC2 Instances and On-Premises Servers with the CloudWatch Agent in the *Amazon CloudWatch User Guide*.	March 28, 2019
SEARCH function added to metric math expressions	You can now use a SEARCH function in metric math expressions. This enables you to create dashboards that update automatically as new resources are created that match the search query. For more information, see Using Search Expressions in Graphs in the *Amazon CloudWatch User Guide*.	March 21, 2019
Amazon SDK Metrics for Enterprise Support	SDK Metrics helps you assess the health of your Amazon services and diagnose latency caused by reaching your account usage limits or by a service outage. For more information, see Monitor Applications Using Amazon SDK Metrics in the *Amazon CloudWatch User Guide*.	December 11, 2018

Alarms on math expressions	CloudWatch supports creating alarms based on metric math expressions. For more information, see Alarms on Math Expressions in the *Amazon CloudWatch User Guide*.	November 20, 2018
New CloudWatch console home page	Amazon has created a new home page in the CloudWatch console, which automatically displays key metrics and alarms for all the Amazon services you are using. For more information, see Getting Started with Amazon CloudWatch in the *Amazon CloudWatch User Guide*.	November 19, 2018
Amazon CloudFormation templates for the CloudWatch Agent	Amazon has uploaded Amazon CloudFormation templates that you can use to install and update the CloudWatch agent. For more information, see Install the CloudWatch Agent on New Instances Using Amazon CloudFormation in the *Amazon CloudWatch User Guide*.	November 9, 2018

Enhancements to the CloudWatch Agent	The CloudWatch agent has been updated to work with both the StatsD and collectd protocols. It also has improved cross-account support. For more informati on, see Retrieve Custom Metrics with StatsD, Retrieve Custom Metrics with collectd, and Sending Metrics and Logs to a Different Amazon Account in the *Amazon CloudWatch User Guide*.	September 28, 2018
Support for Amazon VPC endpoints	You can now establish a private connection between your VPC and CloudWatc h. For more information, see Using CloudWatch with Interface VPC Endpoints in the *Amazon CloudWatch User Guide*.	June 28, 2018

The following table describes important changes to the *Amazon CloudWatch User Guide* before June 2018.

Change	Description	Release date
Metric math	You can now perform math expressions on CloudWatch metrics, producing new time series that you can add to graphs on your dashboard. For more information, see Using math expressions with CloudWatch metrics.	April 4, 2018
"M out of N" alarms	You can now configure an alarm to trigger based on "M out of N" datapoints in any alarm evaluatio	December 8, 2017

Change	Description	Release date
	n interval. For more information, see Evaluating an alarm.	
CloudWatch agent	A new unified CloudWatch agent was released. You can use the unified multi-platform agent to collect custom both system metrics and log files from Amazon EC2 instances and on-premises servers. The new agent supports both Windows and Linux and enables customization of metrics collected, including sub-resource metrics such as per-CPU core. For more information, see Collect metrics, logs, and traces with the CloudWatch agent.	September 7, 2017
NAT gateway metrics	Added metrics for Amazon VPC NAT gateway.	September 7, 2017
High-resolution metrics	You can now optionally set up custom metrics as high-resolution metrics, with a granularity of as low as one second. For more information, see High-resolution metrics.	July 26, 2017
Dashboard APIs	You can now create, modify, and delete dashboards using APIs and the Amazon CLI. For more information, see Creating a customized CloudWatch dashboard.	July 6, 2017
Amazon Direct Connect metrics	Added metrics for Amazon Direct Connect.	June 29, 2017
Amazon VPC VPN metrics	Added metrics for Amazon VPC VPN.	May 15, 2017
AppStream 2.0 metrics	Added metrics for AppStream 2.0.	March 8, 2017

Change	Description	Release date
CloudWatch console color picker	You can now choose the color for each metric on your dashboard widgets. For more information, see Editing a graph on a CloudWatch dashboard .	February 27, 2017
Alarms on dashboards	Alarms can now be added to dashboards. For more information, see Adding an alarm to a CloudWatch dashboard.	February 15, 2017
Added metrics for Amazon Polly	Added metrics for Amazon Polly.	December 1, 2016
Added metrics for Amazon Managed Service for Apache Flink	Added metrics for Amazon Managed Service for Apache Flink.	December 1, 2016
Added support for percentile statistics	You can specify any percentile, using up to two decimal places (for example, p95.45). For more information, see Percentiles.	November 17, 2016
Added metrics for Amazon Simple Email Service	Added metrics for Amazon Simple Email Service.	November 2, 2016
Updated metrics retention	Amazon CloudWatch now retains metrics data for 15 months instead of 14 days.	November 1, 2016
Updated metrics console interface	The CloudWatch console is updated with improvements to existing functionality and new functionality.	November 1, 2016
Added metrics for Amazon Elastic Transcoder	Added metrics for Amazon Elastic Transcoder.	September 20, 2016

Change	Description	Release date
Added metrics for Amazon API Gateway	Added metrics for Amazon API Gateway.	September 9, 2016
Added metrics for Amazon Key Management Service	Added metrics for Amazon Key Management Service.	September 9, 2016
Added metrics for the new Application Load Balancers supported by Elastic Load Balancing	Added metrics for Application Load Balancers.	August 11, 2016
Added new NetworkPa cketsIn and NetworkPa cketsOut metrics for Amazon EC2	Added new NetworkPacketsIn and NetworkPa cketsOut metrics for Amazon EC2.	March 23, 2016
Added new metrics for Amazon EC2 Spot fleet	Added new metrics for Amazon EC2 Spot fleet.	March 21, 2016
Added new CloudWatch Logs metrics	Added new CloudWatch Logs metrics.	March 10, 2016

Change	Description	Release date
Added Amazon OpenSearch Service and Amazon WAF metrics and dimensions	Added Amazon OpenSearch Service and Amazon WAF metrics and dimensions.	October 14, 2015
Added support for CloudWatch dashboards	Dashboards are customizable home pages in the CloudWatch console that you can use to monitor your resources in a single view, even those that are spread out across different Regions. For more information, see Using Amazon CloudWatch dashboards.	October 8, 2015
Added Amazon Lambda metrics and dimensions	Added Amazon Lambda metrics and dimensions.	September 4, 2015
Added Amazon Elastic Container Service metrics and dimensions	Added Amazon Elastic Container Service metrics and dimensions.	August 17, 2015
Added Amazon Simple Storage Service metrics and dimensions	Added Amazon Simple Storage Service metrics and dimensions.	July 26, 2015
New feature: Reboot alarm action	Added the reboot alarm action and new IAM role for use with alarm actions. For more information, see Create alarms to stop, terminate, reboot, or recover an EC2 instance.	July 23, 2015

Change	Description	Release date
Added Amazon WorkSpaces metrics and dimensions	Added Amazon WorkSpaces metrics and dimensions.	April 30, 2015
Added Amazon Machine Learning metrics and dimensions	Added Amazon Machine Learning metrics and dimensions.	April 9, 2015
New feature: Amazon EC2 instance recovery alarm actions	Updated alarm actions to include new EC2 instance recovery action. For more information, see Create alarms to stop, terminate, reboot, or recover an EC2 instance.	March 12, 2015
Added Amazon CloudFront and Amazon CloudSearch metrics and dimensions	Added Amazon CloudFront and Amazon CloudSearch metrics and dimensions.	March 6, 2015
Added Amazon Simple Workflow Service metrics and dimensions	Added Amazon Simple Workflow Service metrics and dimensions.	May 9, 2014
Updated guide to add support for Amazon CloudTrail	Added a new topic to explain how you can use Amazon CloudTrail to log activity in Amazon CloudWatch. For more information, see Logging Amazon CloudWatch API and console operations with Amazon CloudTrail.	April 30, 2014

Change	Description	Release date
Updated guide to use the new Amazon Command Line Interface (Amazon CLI)	The Amazon CLI is a cross-service CLI with a simplified installation, unified configuration, and consistent command line syntax. The Amazon CLI is supported on Linux/Unix, Windows, and Mac. The CLI examples in this guide have been updated to use the new Amazon CLI. For information about how to install and configure the new Amazon CLI, see Getting Set Up with the Amazon CLI Interface in the *Amazon Command Line Interface User Guide*.	February 21, 2014
Added Amazon Redshift and Amazon OpsWorks metrics and dimensions	Added Amazon Redshift and Amazon OpsWorks metrics and dimensions.	July 16, 2013
Added Amazon Route 53 metrics and dimensions	Added Amazon Route 53 metrics and dimensions.	June 26, 2013
New feature: Amazon CloudWatch Alarm Actions	Added a new section to document Amazon CloudWatch alarm actions, which you can use to stop or terminate an Amazon Elastic Compute Cloud instance. For more information, see Create alarms to stop, terminate, reboot, or recover an EC2 instance.	January 8, 2013
Updated EBS metrics	Updated the EBS metrics to include two new metrics for Provisioned IOPS volumes.	November 20, 2012

Change	Description	Release date
New billing alerts	You can now monitor your Amazon charges using Amazon CloudWatch metrics and create alarms to notify you when you have exceeded the specified threshold. For more information, see Create a billing alarm to monitor your estimated Amazon charges.	May 10, 2012
New metrics	You can now access six new Elastic Load Balancing metrics that provide counts of various HTTP response codes.	October 19, 2011
New feature	You can now access metrics from Amazon EMR.	June 30, 2011
New feature	You can now access metrics from Amazon Simple Notification Service and Amazon Simple Queue Service.	July 14, 2011
New Feature	Added information about using the `PutMetricData` API to publish custom metrics. For more information, see Publish custom metrics.	May 10, 2011
Updated metrics retention	Amazon CloudWatch now retains the history of an alarm for two weeks rather than six weeks. With this change, the retention period for alarms matches the retention period for metrics data.	April 7, 2011
New feature	Added ability to send Amazon Simple Notificat ion Service or Auto Scaling notifications when a metric has crossed a threshold. For more information, see Alarms.	December 2, 2010
New feature	A number of CloudWatch actions now include the MaxRecords and NextToken parameters, which enable you to control pages of results to display.	December 2, 2010

Change	Description	Release date
New feature	This service now integrates with Amazon Identity and Access Management (IAM).	December 2, 2010