SDK 版本 3 开发人员指南

Amazon SDK for JavaScript

Copyright © 2024 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon SDK for JavaScript: SDK 版本 3 开发人员指南

Copyright © 2024 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon 的商标和商业外观不得用于任何非 Amazon 的商品或服务，也不得以任何可能引起客户混淆、贬低或诋毁 Amazon 的方式使用。所有非 Amazon 拥有的其他商标均为各自所有者的财产，这些所有者可能附属于 Amazon、与 Amazon 有关联或由 Amazon 赞助，也可能不是如此。

Amazon Web Services 文档中描述的 Amazon Web Services 服务或功能可能因区域而异。要查看适用于中国区域的差异，请参阅 中国的 Amazon Web Services 服务入门 (PDF)。

Table of Contents

.... viii
Amazon SDK for JavaScript 是什么？ 1
SDK 主要版本的维护和支持 1
版本 3 中的新增功能 2
模块化软件包 2
新的中间件堆栈 6
将 SDK 与 Node.js 结合使用 7
将 SDK 与 Amazon Cloud9 结合使用 7
将 SDK 与 Amazon Amplify 结合使用 7
将 SDK 与 Web 浏览器结合使用 8
在 V3 中使用浏览器 8
常见使用案例 8
关于示例 9
资源 9
开始使用 10
使用 Amazon 进行 SDK 身份验证 10
开始 Amazon 访问门户会话 11
更多身份验证信息 12
开始使用 Node.js 12
情景 13
先决条件 13
步骤 1：设置软件包结构并安装客户端程序包 13
步骤 2：添加必要的导入和 SDK 代码 14
步骤 3：运行示例 16
开始使用浏览器 16
情景 17
步骤 1：创建一个 Amazon Cognito 身份池和 IAM 角色 17
步骤 2：将策略添加到创建的 IAM 角色 18
步骤 3：添加 Amazon S3 存储桶和对象 19
步骤 4：设置浏览器代码 20
步骤 5：运行示例 20
清理 21
将 SDK 设置为 JavaScript 22
先决条件 22

设置 Amazon Node.js 环境 ... 22
支持的 Web 浏览器 ... 23
安装 SDK ... 24
加载软件开发工具包 ... 24
将 SDK 配置为 JavaScript ... 25
每个服务的配置 ... 25
为每项服务设置配置 ... 26
设置 Amazon 区域 ... 26
在客户端类构造函数中 ... 26
使用环境变量 ... 26
使用共享配置文件 ... 27
设置区域的优先顺序 ... 27
设置凭据 ... 27
凭证的最佳实践 ... 28
在 Node.js 中设置凭据 ... 28
在 Web 浏览器中设置凭据 ... 31
Node.js 注意事项 ... 34
使用内置的 Node.js 模块 ... 34
使用 npm 软件包 ... 35
在 Node.js 中配置 maxSockets ... 35
在 Node.js 中使用保持活动状态重用连接 ... 36
为 Node.js 配置代理 ... 36
在 Node.js 中注册证书包 ... 37
浏览器脚本注意事项 ... 38
为浏览器构建 SDK ... 38
跨源资源共享 (CORS) ... 39
与 webpack 捆绑包 ... 42
使用 Amazon 服务 ... 47
创建和调用服务对象 ... 47
指定服务对象参数 ... 48
异步呼叫服务 ... 48
管理异步调用 ... 49
使用异步/等待 ... 49
使用承诺 ... 50
使用回调函数 ... 51
创建服务客户端请求 ... 52

- 处理服务客户端的响应 ... 53
  - 访问响应中返回的数据 ... 53
  - 访问错误信息 ... 53
- 使用 JSON ... 54
  - 将 JSON 作为服务对象参数 ... 54
- 包含指导的代码示例子集 ... 55
  - JavaScript ES6/CommonJS 语法 ... 56
  - Amazon DynamoDB 示例 ... 59
  - AWS Elemental MediaConvert 示例 ... 82
  - Amazon Lambda 示例 ... 103
  - Amazon Lex 示例 ... 104
  - Amazon Polly 示例 ... 104
  - Amazon Redshift 示例 ... 107
  - Amazon SES 示例 ... 114
  - Amazon SNS 示例 ... 140
  - Amazon Transcribe 示例 ... 172
  - 跨服务：在 Amazon EC2 实例上设置 Node.js ... 183
  - 跨服务：用于提交数据的应用程序 ... 185
  - 跨服务：转录应用程序 ... 193
  - 跨服务：Amazon API Gateway 和 Lambda ... 204
  - 跨服务：使用 Step Functions 的无服务器工作流 ... 219
  - 跨服务：计划的 Lambda 事件 ... 233
  - 跨服务：Amazon Lex 示例 ... 244
  - 跨服务：消息收发应用程序 ... 258

与 SDK Amazon Cloud9 一起使用适用于 JavaScript ... 271

- 第 1 步：设置要使用的 Amazon 账户 Amazon Cloud9 ... 271
- 第 2 步：设置 Amazon Cloud9 开发环境 ... 271
- 步骤 3：设置 SDK JavaScript ... 272
  - 为 Node.js 设置软件开发 JavaScript 工具包 ... 272
  - 在浏览器 JavaScript 中设置 SDK ... 273
- 步骤 4：下载示例代码 ... 273
- 步骤 5：运行并调试示例代码 ... 273

代码示例 ... 274

- 操作和场景 ... 274
  - Auto Scaling ... 275
  - Amazon Bedrock ... 318

Amazon Bedrock 运行时系统 ... 323
Agents for Amazon Bedrock ... 338
亚马逊 Bedrock Runtime 的代理 ... 352
CloudWatch ... 354
CloudWatch 活动 ... 370
CloudWatch 日志 ... 377
CodeBuild ... 394
Amazon Cognito 身份提供者 ... 397
DynamoDB ... 417
Amazon EC2 ... 463
Elastic Load Balancing ... 545
EventBridge ... 594
Amazon Glue ... 601
HealthImaging ... 624
IAM ... 661
Lambda ... 771
Amazon Personalize ... 781
Amazon Personalize Events ... 798
Amazon Personalize Runtime ... 802
Amazon Pinpoint ... 806
Amazon Redshift ... 816
Amazon S3 ... 821
S3 Glacier ... 858
SageMaker ... 862
Secrets Manager ... 895
Amazon SES ... 897
Amazon SNS ... 920
Amazon SQS ... 959
Step Functions ... 995
Amazon STS ... 997
Amazon Web Services Support ... 1000
Amazon Transcribe ... 1018
跨服务示例 ... 1027
构建 Amazon Transcribe 应用程序 ... 1027
构建 Amazon Transcribe 流式传输应用程序 ... 1028
构建应用程序以将数据提交到 DynamoDB 表 ... 1028

构建 Amazon Lex 聊天机器人 1029
创建无服务器应用程序来管理照片 1029
创建 Web 应用程序来跟踪 DynamoDB 数据 1030
创建 Aurora Serverless 工作项跟踪器 1030
创建 Amazon Textract 浏览器应用程序 1031
创建用于分析客户反馈的应用程序 1031
检测图像中的 PPE 1035
检测图像中的对象 1036
检测视频中的人物和对象 1036
从浏览器调用 Lambda 函数 1037
使用 API Gateway 调用 Lambda 函数 1038
使用 Step Functions 调用 Lambda 函数 1038
使用计划的事件调用 Lambda 函数 1039
安全性 1040
数据保护 1040
Identity and Access Management 1041
受众 1041
使用身份进行身份验证 1042
使用策略管理访问 1044
如何 Amazon Web Services 使用 IAM 1046
对 Amazon 身份和访问进行故障排除 1046
合规性验证 1048
韧性 1048
基础设施安全性 1049
强制使用最低版本的 TLS 1049
在 Node.js 中验证并强制执行 TLS 1050
在浏览器脚本中验证并强制执行 TLS 1052
迁移到版本 3 1054
迁移到 V3 1054
使用 codemod 迁移现有的 v2 代码 1054
文档历史记录 1056
文档历史记录 1056

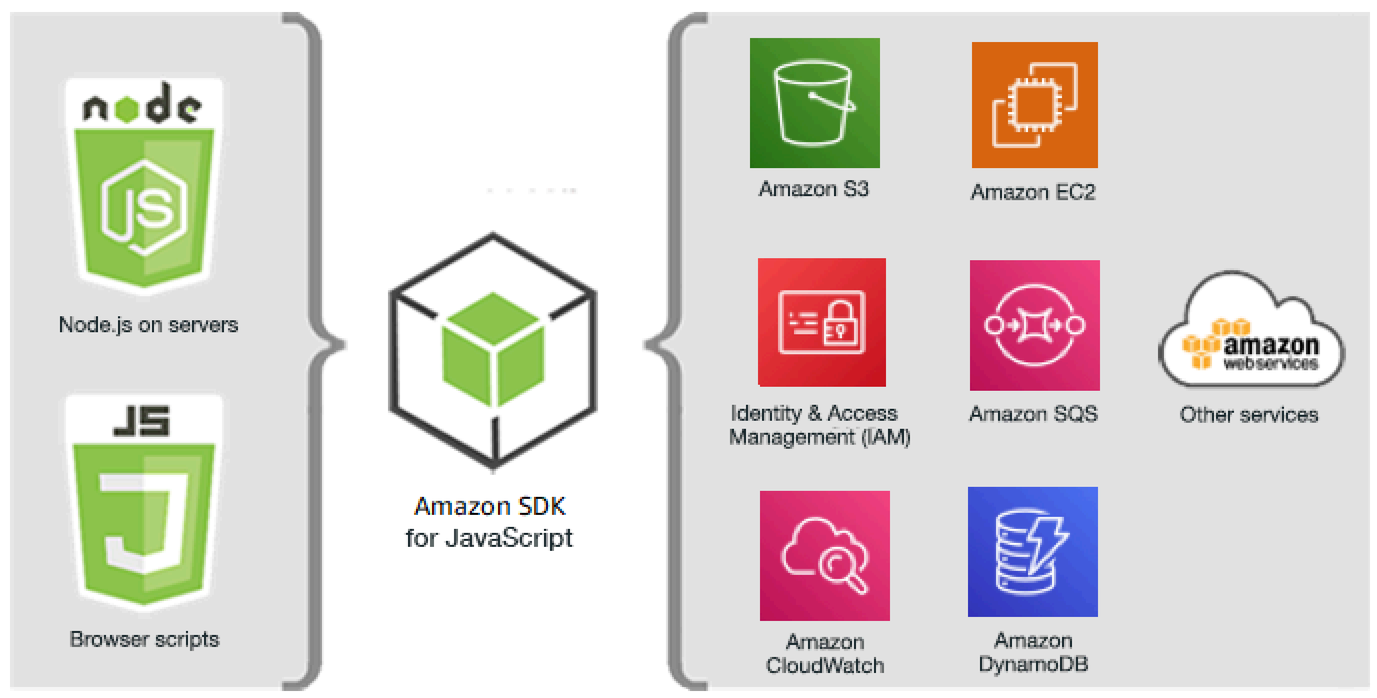
Amazon SDK for JavaScript V3 API 参考指南详细描述了 Amazon SDK for JavaScript 版本 3 (V3) 的所有 API 操作。

本文属于机器翻译版本。若本译文内容与英语原文存在差异，则一律以英文原文为准。

Amazon SDK for JavaScript 是什么？

欢迎阅读《Amazon SDK for JavaScript 开发人员指南》。本指南提供有关设置和配置 Amazon SDK for JavaScript 的一般信息。它还会引导您完成使用 Amazon SDK for JavaScript 运行各种 Amazon 服务的示例和教程。

Amazon SDK for JavaScript v3 API 参考指南为 Amazon 服务提供了 JavaScript API。您可以使用 JavaScript API 构建适用于 Node.js 或浏览器的库或应用程序。



SDK 主要版本的维护和支持

有关维护和支持 SDK 主要版本及其基础依赖关系的信息，请参阅 Amazon SDK 和工具参考指南中的以下内容：

- Amazon SDK 和工具维护策略
- Amazon SDK 和工具版本支持矩阵

版本 3 中的新增功能

SDK for JavaScript 版本 3 (V3) 包含以下新功能。

模块化软件包

用户现在可以为每项服务使用单独的软件包。

新的中间件堆栈

用户现在可以使用中间件堆栈来控制操作调用的生命周期。

此外，该 SDK 是用 TypeScript 编写的，具有许多优点，例如静态输入。

> **Important**
>
> 本指南中 V3 的代码示例是用 ECMAScript 6 (ES6) 编写的。ES6 带来了新的语法和新功能，使您的代码更现代、更具可读性，并能做到更多的事情。要使用 ES6，您需要使用 Node.js 版本 13.x 或更高版本。要下载并安装最新版本的 Node.js，请参阅 Node.js 下载。有关更多信息，请参阅JavaScript ES6/CommonJS 语法。

模块化软件包

SDK for JavaScript 版本 2 (V2) 要求您使用整个 Amazon SDK，如下所示。

```
var AWS = require("aws-sdk");
```

如果您的应用程序使用许多 Amazon 服务，那么加载整个 SDK 不是问题。但是，如果您只需要使用少量 Amazon 服务，则意味着需要使用不需要或不使用的代码来增加应用程序的大小。

在 V3 中，您只能加载和使用所需的各项 Amazon 服务。以下示例显示了这一点，通过它您可以访问 Amazon DynamoDB (DynamoDB)。

```
import { DynamoDB } from "@aws-sdk/client-dynamodb";
```

您不仅可以加载和使用各项 Amazon 服务，还可以仅加载和使用所需的服务命令。以下示例显示了这一点，通过它您可以访问 DynamoDB 客户端和 `ListTablesCommand` 命令。

```
import {
```

```
  DynamoDBClient,
  ListTablesCommand
} from "@aws-sdk/client-dynamodb";
```

> **Important**
>
> 不应将子模块导入模块中。例如，以下代码可能会导致错误：
>
> ```
> import { CognitoIdentity } from "@aws-sdk/client-cognito-identity/
> CognitoIdentity";
> ```
>
> 以下是正确的代码。
>
> ```
> import { CognitoIdentity } from "@aws-sdk/client-cognito-identity";
> ```

比较代码大小

在版本 2 (V2) 中，列出您在 `us-west-2` 区域的所有 Amazon DynamoDB 表的简单代码示例可能如下所示。

```
var AWS = require("aws-sdk");
// Set the Region
AWS.config.update({region: "us-west-2"});
// Create DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

// Call DynamoDB to retrieve the list of tables
ddb.listTables({ Limit:10 }, function(err, data) {
  if (err) {
    console.log("Error", err.code);
  } else {
    console.log("Tables names are ", data.TableNames);
  }
});
```

V3 如下所示。

```
import {
  DynamoDBClient,
```

```
  ListTablesCommand
} from "@aws-sdk/client-dynamodb";
(async function () {
   const dbclient = new DynamoDBClient({ region: 'us-west-2'});

  try {
    const results = await dbclient.send(new ListTablesCommand);
    results.TableNames.forEach(function (item, index) {
      console.log(item);
    });
  } catch (err) {
    console.error(err)
  }
})();
```

aws-sdk 软件包会为您的应用程序增加大约 40 MB。将 var AWS = require("aws-sdk") 替换为 import {DynamoDB} from "@aws-sdk/client-dynamodb" 可将开销减少到大约 3 MB。将导入限制为仅 DynamoDB 客户端和 ListTablesCommand 命令，可将开销减少到 100 KB 以下。

```
// Load the DynamoDB client and ListTablesCommand command for Node.js
import {
  DynamoDBClient,
  ListTablesCommand
} from "@aws-sdk/client-dynamodb";
const dbclient = new DynamoDBClient({});
```

在 V3 中调用命令

您可以使用 V2 或 V3 命令在 V3 中执行操作。要使用 V3 命令，请导入命令和所需的 Amazon 服务包客户端，然后使用异步/等待模式通过 .send 方法运行命令。

要使用 V2 命令，您需要导入所需的 Amazon 服务包，然后使用回调或异步/等待模式直接在软件包中运行 V2 命令。

使用 V3 命令

V3 为每个 Amazon 服务包提供了一组命令，使您能够对该 Amazon 服务执行操作。安装 Amazon 服务后，您可以浏览项目中 node-modules/@aws-sdk/client-*PACKAGE_NAME*/commands folder. 的可用命令

您必须导入要使用的命令。例如，以下代码可加载 DynamoDB 服务和 CreateTableCommand 命令。

```
import { DynamoDB, CreateTableCommand } from "@aws-sdk/client-dynamodb";
```

要以推荐的异步/等待模式调用这些命令，请使用以下语法。

```
CLIENT.send(new XXXCommand)
```

例如，以下示例使用推荐的异步/等待模式创建 DynamoDB 表。

```
import { DynamoDB, CreateTableCommand } from "@aws-sdk/client-dynamodb";
const dynamodb = new DynamoDB({region: 'us-west-2'});
var tableParams = {
    Table : TABLE_NAME
};
(async function () => {
      try{
           const data = await dynamodb.send(new CreateTableCommand(tableParams));
           console.log("Success", data);
      }
      catch (err) {
           console.log("Error", err);
      }
})();
```

使用 V2 命令

要在 SDK for JavaScript 中使用 V2 命令，您需要导入完整的 Amazon 服务包，如以下代码所示。

```
const { DynamoDB } = require('@aws-sdk/client-dynamodb');
```

要以推荐的异步/等待模式调用 V2 命令，请使用以下语法。

```
client.command(parameters)
```

以下示例使用 V2 `createTable` 命令，通过推荐的异步/等待模式创建 DynamoDB 表。

```
const {DynamoDB} = require('@aws-sdk/client-dynamodb');
const dymamoDB = new DynamoDB({region: 'us-west-2'});
var tableParams = {
    TableName : TABLE_NAME
};
```

```
async function run() => {
      try {
           const data = await dymamoDB.createTable(tableParams);
           console.log("Success", data);
      }
      catch (err) {
           console.log("Error", err);
      }
};
run();
```

以下示例使用 V2 createBucket 命令，通过回调模式创建 Amazon S3 存储桶。

```
const {S3} = require('@aws-sdk/client-s3');
const s3 = new S3({region: 'us-west-2'});
var bucketParams = {
    Bucket : BUCKET_NAME
};
function run(){
         s3.createBucket(bucketParams, function(err, data) {
         if (err) {
         console.log("Error", err);
         } else {
         console.log("Success", data.Location);
         }
    })
};
```

新的中间件堆栈

借助 SDK V2，您可以通过为请求附加事件侦听器，在请求生命周期的多个阶段修改请求。这种方法可能使调试请求生命周期中的错误变得困难。

在 V3 中，您可以使用新的中间件堆栈来控制操作调用的生命周期。这种方法有几个好处。堆栈中的每个中间件阶段都会在对请求对象进行任何更改后调用下一个中间件阶段。这也使调试堆栈中的问题变得更加容易，因为您可以准确地看到哪些被调用的中间件阶段导致了错误。

以下示例使用中间件向 Amazon DynamoDB 客户端（我们之前创建并演示了该客户端）添加自定义标头。第一个参数是一个接受 next 的函数，该函数指要调用的堆栈中的下一个中间件阶段，还有 context，这是一个包含有关正在调用的操作的一些信息的对象。该函数返回一个接受 args 的函数，这是一个包含传递给操作和请求的参数的对象。它使用 args 调用下一个中间件，然后返回结果。

```
dbclient.middlewareStack.add(
  (next, context) => args => {
    args.request.headers["Custom-Header"] = "value";
    return next(args);
  },
  {
    step: "build"
  }
);

dbclient.send(new PutObjectCommand(params));
```

将 SDK 与 Node.js 结合使用

Node.js 是一个用于运行服务器端 JavaScript 应用程序的跨平台运行时系统。您可以在 Amazon Elastic Compute Cloud (Amazon EC2) 实例上设置 Node.js 以在服务器上运行。您还可以使用 Node.js 来编写按需 Amazon Lambda 函数。

使用 SDK for Node.js 与在 Web 浏览器中将其用于 JavaScript 的方式不同。区别在于，您加载 SDK 以及获取访问特定 Web 服务所需凭证的方法有所不同。如果在 Node.js 与浏览器之间使用特定 API 存在差别时，我们将对这些差别进行说明。

将 SDK 与 Amazon Cloud9 结合使用

您还可以使用在 Amazon Cloud9 IDE 中使用 SDK for JavaScript 开发 Node.js 应用程序。有关将 Amazon Cloud9 与 SDK for JavaScript 结合使用的更多信息，请参阅Amazon Cloud9 搭配使用 Amazon SDK for JavaScript。

将 SDK 与 Amazon Amplify 结合使用

对于基于浏览器的 Web、移动和混合应用程序，您也可以使用 GitHub 上的 Amazon Amplify 库。它扩展了 SDK for JavaScript，提供了一个声明性接口。

> **Note**
>
> Amplify 等框架可能无法提供与 SDK for JavaScript 相同的浏览器支持。有关详细信息，请查看框架文档。

将 SDK 与 Web 浏览器结合使用

所有主流 Web 浏览器支持 JavaScript 的执行。在 Web 浏览器中运行的 JavaScript 代码通常称为客户端 JavaScript。

有关 Amazon SDK for JavaScript 支持的浏览器列表，请参阅支持的 Web 浏览器。

在 Web 浏览器中使用 SDK for JavaScript 与在 Node.js 中使用它的方式不同。区别在于，您加载 SDK 以及获取访问特定 Web 服务所需凭证的方法有所不同。如果在 Node.js 与浏览器之间使用特定 API 存在差别时，我们将对这些差别进行说明。

在 V3 中使用浏览器

V3 允许您仅将所需的 SDK for JavaScript 文件捆绑和包含在浏览器中，从而减少开销。

要在 HTML 页面中使用 SDK for JavaScript V3，必须使用 Webpack 将所需的客户端模块和所有必需的 JavaScript 函数捆绑到一个 JavaScript 文件中，然后将其添加到 HTML 页面 <head> 的脚本标签中。例如：

```
<script src="./main.js"></script>
```

有关 Webpack 的更多信息，请参阅将应用程序与 webpack 捆绑在一起。

要使用 SDK for JavaScript V2，您需要添加一个指向 V2 SDK 最新版本的脚本标签。有关更多信息，请参阅《Amazon SDK for JavaScript 开发人员指南 v2》中的示例。

常见使用案例

在浏览器脚本中使用 SDK for JavaScript 实现了多种颇具吸引力的使用案例。通过使用 SDK for JavaScript 访问各种 Web 服务，您可以在浏览器应用程序中构建一些东西，此处介绍了几个相关想法。

- 构建 Amazon 服务的自定义控制台，在其中您可以跨区域和服务访问并组合功能，从而最好地满足您的组织或项目需求。
- 使用 Amazon Cognito 以启用对您的浏览器应用程序和网站的经身份验证用户的访问，包括使用来自 Facebook 和其他提供商的第三方身份验证。

- 使用 Amazon Kinesis 实时处理点击流或其他营销数据。
- 为无服务器数据持久性使用 Amazon DynamoDB，例如针对网站访问者或应用程序用户的单独用户首选项。
- 使用 Amazon Lambda 封装专有逻辑，您可以从浏览器脚本调用逻辑而无需下载和向用户泄露您的知识产权。

关于示例

您可以在 Amazon 代码示例存储库中浏览 SDK for JavaScript 示例。

资源

除了本指南外，还有以下适用于 SDK for JavaScript 开发人员的在线资源：

- Amazon SDK for JavaScript V3 API 参考指南
- Amazon SDK 和工具参考指南：包含 Amazon SDK 中常见的设置、功能和其他基础概念。
- JavaScript 开发人员博客
- Amazon JavaScript 论坛
- Amazon 代码目录中的 JavaScript 示例
- Amazon 代码示例存储库
- Gitter 通道
- 堆栈溢出
- 堆栈溢出问题 taggedAWS -sdk-js
- GitHub
  - SDK 源
  - 文档源

开始使用 Amazon SDK for JavaScript

Amazon SDK for JavaScript 提供了在浏览器或 Node.js 中对 Web 服务的访问。本部分有两个入门练习，向您演示了如何在每个 JavaScript 环境中使用 SDK for JavaScript。

> **Note**
>
> 您可以在 Amazon Cloud9 IDE 中使用 SDK for JavaScript 开发 Node.js 应用程序，为基于浏览器的应用程序开发 JavaScript。有关如何使用 Amazon Cloud9 for Node.js 开发的示例，请参阅Amazon Cloud9 搭配使用 Amazon SDK for JavaScript。

主题

- 使用 Amazon 进行 SDK 身份验证
- 开始使用 Node.js
- 开始使用浏览器

使用 Amazon 进行 SDK 身份验证

使用 Amazon Web Services进行开发时，您必须确定您的代码是如何使用 Amazon 进行身份验证的。您可以通过不同方式配置对 Amazon 资源的编程访问权限，具体取决于环境和可用的 Amazon 访问权限。

要选择您的身份验证方法并针对 SDK 进行配置，请参阅《Amazon SDK 和工具参考指南》中的身份验证和访问。

我们建议在本地开发且雇主未向其提供身份验证方法的新用户对 Amazon IAM Identity Center 进行设置。此方法包括安装 Amazon CLI 以便于配置和定期登录 Amazon 访问门户。如果您选择此方法，则在完成《Amazon SDKs and Tools Reference Guide》中的 IAM Identity Center authentication 程序后，您的环境应包含以下元素：

- Amazon CLI，您可用于在运行应用程序之前启动 Amazon 访问门户会话。
- 共享 Amazon`config` 文件，其 `[default]` 配置文件包含一组可从 SDK 中引用的配置值。要查找此文件的位置，请参阅《Amazon SDK 和工具参考指南》中的共享文件的位置。
- 该共享 `config` 文件设置了 `region` 设置。这将设置 SDK 用于 Amazon 请求的默认 Amazon Web Services 区域。此区域用于未指定要使用的区域的 SDK 服务请求。

- 在向 Amazon 发送请求之前，SDK 使用配置文件的 SSO 令牌提供程序配置来获取凭证。sso_role_name 值是与 IAM Identity Center 权限集关联的 IAM 角色，允许访问应用程序中使用的 Amazon Web Services。

  以下示例 config 文件展示了使用 SSO 令牌提供程序配置进行设置的默认配置文件。配置文件的 sso_session 设置引用所指定的 sso-session 部分。sso-session 节包含启动 Amazon 访问门户会话的设置。

  ```
  [default]
  sso_session = my-sso
  sso_account_id = 111122223333
  sso_role_name = SampleRole
  region = us-east-1
  output = json

  [sso-session my-sso]
  sso_region = us-east-1
  sso_start_url = https://provided-domain.awsapps.com/start
  sso_registration_scopes = sso:account:access
  ```

Amazon SDK for JavaScript v3 无需向应用程序添加其他包（例如 SSO 和 SSOOIDC）即可使用 IAM Identity Center 身份验证。

有关明确使用此凭证提供程序的详细信息，请参阅 npm（Node.js 程序包管理器）网站上的 fromSSO()。

开始 Amazon 访问门户会话

在运行访问 Amazon Web Services 的应用程序之前，需要有活动的 Amazon 访问门户会话，以便 SDK 使用 IAM Identity Center 身份验证来解析凭证。根据配置的会话时长，访问权限最终将过期，并且 SDK 将遇到身份验证错误。要登录 Amazon 访问门户，请在 Amazon CLI 中运行以下命令。

```
aws sso login
```

如果遵循指南并具有默认的配置文件设置，则无需使用 --profile 选项来调用该命令。如果您的 SSO 令牌提供程序配置在使用指定的配置文件，则命令为 aws sso login --profile named-profile。

（可选）要测试是否已有活动会话，请运行以下 Amazon CLI 命令。

```
aws sts get-caller-identity
```

如果会话是活动的，则对此命令的响应会报告共享 `config` 文件中配置的 IAM Identity Center 账户和权限集。

> **Note**
>
> 如果您已经有一个有效的 Amazon 访问门户会话并且运行了 `aws sso login`，则无需提供凭证。
> 登录过程可能会提示您允许 Amazon CLI 访问您的数据。由于 Amazon CLI 基于 SDK for Python 构建，因此权限消息可能包含 `botocore` 名称的变体。

更多身份验证信息

人类用户，也称为人类身份，是应用程序的人员、管理员、开发人员、操作员和使用者。他们必须有身份才能访问您的 Amazon 环境和应用程序。作为组织成员的人类用户（即您、开发人员）也称为工作人员身份。

访问 Amazon 时使用临时凭证。您可以使用身份提供商来以担任角色的形式提供为人类用户对 Amazon 账户的联合访问权限，这将提供临时证书。对于集中式访问权限管理，我们建议使用 Amazon IAM Identity Center (IAM Identity Center) 来管理对您账户的访问权限以及这些账户中的其它权限。有关更多替代方案，请参阅以下内容：

- 有关最佳实践的更多信息，请参阅《IAM 用户指南》中的 IAM 中的安全最佳实践。
- 要创建短期 Amazon 凭证，请参阅《IAM 用户指南》中的临时安全凭证。
- 要了解其他 Amazon SDK for JavaScript V3 凭证提供程序，请参阅《Amazon SDK 和工具参考指南》中的标准化凭证提供程序。

开始使用 Node.js

本指南向您演示了如何初始化 NPM 软件包、向软件包中添加服务客户端以及如何使用 JavaScript SDK 调用服务操作。

情景

使用一个执行以下操作的主文件创建一个新的 NPM 软件包：

- 创建 Amazon Simple Storage Service 存储桶
- 将对象放入 Amazon S3 存储桶
- 读取 Amazon S3 存储桶中的对象
- 确认用户是否要删除资源

先决条件

在运行示例之前，您必须先执行以下操作：

- 配置 SDK 身份验证。有关更多信息，请参阅使用 Amazon 进行 SDK 身份验证。
- 安装 Node.js。

步骤 1：设置软件包结构并安装客户端程序包

设置程序包结构并安装客户端程序包：

1. 创建一个新文件夹 `nodegetstarted` 用于包含程序包。
2. 从命令行导航到新文件夹。
3. 运行以下命令以创建默认的 `package.json` 文件：

```
npm init -y
```

4. 要安装 Amazon S3 客户端程序包，请运行以下命令：

```
npm i @aws-sdk/client-s3
```

5. 将 `"type": "module"` 添加到 `package.json` 文件。这会告诉 Node.js 使用现代 ESM 语法。最终的 `package.json` 应类似于以下内容：

```
{
  "name": "example-javascriptv3-get-started-node",
  "version": "1.0.0",
```

```
  "description": "This guide shows you how to initialize an NPM package, add a
 service client to your package, and use the JavaScript SDK to call a service
 action.",
  "main": "index.js",
  "scripts": {
"test": "vitest run **/*.unit.test.js"
  },
  "author": "Your Name"
  "license": "Apache-2.0",
  "dependencies": {
 "@aws-sdk/client-s3": "^3.420.0"
  },
  "type": "module"
}
```

步骤 2：添加必要的导入和 SDK 代码

将以下代码添加到 nodegetstarted 文件夹中名为 index.js 的文件中。

```
// This is used for getting user input.
import { createInterface } from "readline/promises";

import {
  S3Client,
  PutObjectCommand,
  CreateBucketCommand,
  DeleteObjectCommand,
  DeleteBucketCommand,
  paginateListObjectsV2,
  GetObjectCommand,
} from "@aws-sdk/client-s3";

export async function main() {
  // A region and credentials can be declared explicitly. For example
  // `new S3Client({ region: 'us-east-1', credentials: {...} })` would
  //initialize the client with those settings. However, the SDK will
  // use your local configuration and credentials if those properties
  // are not defined here.
  const s3Client = new S3Client({});

  // Create an Amazon S3 bucket. The epoch timestamp is appended
```

```
  // to the name to make it unique.
  const bucketName = `test-bucket-${Date.now()}`;
  await s3Client.send(
    new CreateBucketCommand({
      Bucket: bucketName,
    })
  );

  // Put an object into an Amazon S3 bucket.
  await s3Client.send(
    new PutObjectCommand({
      Bucket: bucketName,
      Key: "my-first-object.txt",
      Body: "Hello JavaScript SDK!",
    })
  );

  // Read the object.
  const { Body } = await s3Client.send(
    new GetObjectCommand({
      Bucket: bucketName,
      Key: "my-first-object.txt",
    })
  );

  console.log(await Body.transformToString());

  // Confirm resource deletion.
  const prompt = createInterface({
    input: process.stdin,
    output: process.stdout,
  });

  const result = await prompt.question("Empty and delete bucket? (y/n) ");
  prompt.close();

  if (result === "y") {
    // Create an async iterator over lists of objects in a bucket.
    const paginator = paginateListObjectsV2(
      { client: s3Client },
      { Bucket: bucketName }
    );
    for await (const page of paginator) {
      const objects = page.Contents;
```

```
      if (objects) {
        // For every object in each page, delete it.
        for (const object of objects) {
          await s3Client.send(
            new DeleteObjectCommand({ Bucket: bucketName, Key: object.Key })
          );
        }
      }
    }

    // Once all the objects are gone, the bucket can be deleted.
    await s3Client.send(new DeleteBucketCommand({ Bucket: bucketName }));
  }
}

// Call a function if this file was run directly. This allows the file
// to be runnable without running on import.
import { fileURLToPath } from "url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main();
}
```

此示例代码可在 GitHub 上的此处找到。

步骤 3：运行示例

> **Note**
>
> 请记得登录！如果您使用 IAM Identity Center，请记住使用 Amazon CLI `aws sso login` 命令登录。

1. 运行 `node index.js`。
2. 选择是否清空并删除存储桶。
3. 如果您不删除存储桶，请务必手动清空并稍后将其删除。

开始使用浏览器

本节将向您介绍一个示例，该示例演示了如何在浏览器中运行 SDK for JavaScript 版本 3 (V3)。

> **Note**
>
> 在浏览器中运行 V3 与版本 2 (V2) 略有不同。有关更多信息，请参阅在 V3 中使用浏览器。

有关使用 SDK for JavaScript (V3) 的其他示例，请参阅 适用于 JavaScript (v3) 代码示例的 SDK。

此 Web 应用程序示例向您展示：

- 如何使用 Amazon Cognito 进行身份验证以访问 Amazon 服务。
- 如何使用 Amazon Identity and Access Management (IAM) 角色读取 Amazon Simple Storage Service (Amazon S3) 存储桶中的对象列表。

> **Note**
>
> 此示例不使用 Amazon IAM Identity Center 进行身份验证。

情景

Amazon S3 是一项对象存储服务，提供行业领先的可扩展性、数据可用性、安全性和性能。您可以使用 Amazon S3 将数据作为对象存储在名为存储桶的容器中。有关 Amazon S3 的更多信息，请参阅 Amazon S3 用户指南。

此示例向您演示了如何设置和运行代入 IAM 角色的 Web 应用程序，以便从 Amazon S3 存储桶中进行读取。该示例使用 React 前端库和 Vite 前端工具来提供 JavaScript 开发环境。该 Web 应用程序使用 Amazon Cognito 身份池来提供访问 Amazon 服务所需的凭证。随附的代码示例演示了在 Web 应用程序中加载和使用 SDK for JavaScript 的基本模式。

步骤 1：创建一个 Amazon Cognito 身份池和 IAM 角色

在本练习中，您将创建并使用一个 Amazon Cognito 身份池，为 Web 应用程序提供对 Amazon S3 服务的无需验证身份的访问权限。创建身份池还会创建一个 Amazon Identity and Access Management (IAM) 角色来支持未经身份验证的来宾用户。在本练习中，我们仅使用未经身份验证的用户角色，将重点放在任务上。您可在以后集成对身份提供商和通过身份验证的用户的支持。有关添加 Amazon Cognito 身份池的更多信息，请参阅《Amazon Cognito 开发人员指南》中的教程：创建身份池。

创建一个 Amazon Cognito 身份池和关联的 IAM 角色

1. 登录 Amazon Web Services Management Console 并打开 Amazon Cognito 控制台，网址为 https://console.aws.amazon.com/cognito/。
2. 在左侧导航窗格中，选择身份池。
3. 选择创建身份池。
4. 在配置身份池信任中，选择访客访问权限进行用户身份验证。
5. 在配置权限中，选择创建新的 IAM 角色并在 IAM 角色名称中输入一个名称（例如 getStartedRole）。
6. 在配置属性中，在身份池名称中输入一个名称（例如 getStartedPool）。
7. 在查看并创建中，确认您为新身份池所做的选择。选择编辑以返回向导并更改任何设置。完成后，选择创建身份池。
8. 记下新创建的 Amazon Cognito 身份池的身份池 ID 和区域。您需要这些值以便替换 步骤 4：设置浏览器代码 中的*IDENTITY_POOL_ID* 和 *REGION*。

在创建 Amazon Cognito 身份池之后，您已准备好添加 Web 应用程序所需的 Amazon S3 的权限。

步骤 2：将策略添加到创建的 IAM 角色

要允许访问您的 Web 应用程序中的 Amazon S3 存储桶，请使用为您的 Amazon Cognito 身份池（例如 getStartedPool）创建的未经身份验证的 IAM 角色（例如 getStartedPool）。这需要您将 IAM 策略添加到角色。有关修改 IAM 角色的更多信息，请参阅《IAM 用户指南》中的修改角色权限策略。

将 Amazon S3 策略添加到与未经身份验证用户关联的 IAM 角色

1. 登录 Amazon Web Services Management Console，然后通过以下网址打开 IAM 控制台：https://console.aws.amazon.com/iam/。
2. 在左侧导航窗格中，选择角色。
3. 选择要修改的角色的名称（例如，getStartedRole），然后选择权限选项卡。
4. 选择添加权限，然后选择附加策略。
5. 在此角色的添加权限页面中，找到并选中 AmazonS3ReadOnlyAccess 的复选框。

> Note
>
> 您可以使用此流程来启用对任何 Amazon 服务的访问权限。

6. 选择添加权限。

在您创建 Amazon Cognito 身份池并将 Amazon S3 的权限添加到未验证身份用户的 IAM 角色之后，您已准备好添加并配置 Amazon S3 存储桶。

步骤 3：添加 Amazon S3 存储桶和对象

在此步骤中，您将为示例添加 Amazon S3 存储桶和对象。您还将启用存储桶的跨源资源共享 (CORS)。有关创建 Amazon S3 存储桶和对象的更多信息，请参阅《Amazon S3 入门指南》中的 Amazon S3 入门。

使用 CORS 添加 Amazon S3 存储桶和对象

1. 登录到 Amazon Web Services Management Console，然后通过以下网址打开 Amazon S3 控制台：https://console.aws.amazon.com/s3/。
2. 在左侧的导航窗格中，选择存储桶，然后选择创建存储桶。
3. 输入符合存储桶命名规则的存储桶名称（例如 getstartedbucket），然后选择创建存储桶。
4. 选择您创建的存储桶，然后选择对象选项卡。然后选择上传。
5. 在文件和文件夹下，选择添加文件。
6. 选择要上传的文件，然后选择打开。然后选择上传以完成将对象上传到您的存储桶。
7. 接下来，选择存储桶的权限选项卡，然后在跨源资源共享 (CORS) 部分选择编辑。输入以下 JSON：

```
[
  {
    "AllowedHeaders": [
      "*"
    ],
    "AllowedMethods": [
      "GET"
    ],
    "AllowedOrigins": [
      "*"
    ],
    "ExposeHeaders": []
  }
]
```

8. 选择保存更改。

添加 Amazon S3 存储桶并添加对象后，您就可以设置浏览器代码了。

步骤 4：设置浏览器代码

示例应用程序包含一个单页的 React 应用程序。此示例的文件可以在 GitHub 上的此处找到。

设置示例应用程序

1. 安装 Node.js。
2. 从命令行中克隆 Amazon 代码示例存储库：

```
git clone --depth 1 https://github.com/awsdocs/aws-doc-sdk-examples.git
```

3. 导航到示例应用程序：

```
cd aws-doc-sdk-examples/javascriptv3/example_code/web/s3/list-objects/
```

4. 要安装所需的程序包，请运行以下命令：

```
npm install
```

5. 接下来，在文本编辑器中打开 `src/App.tsx` 并完成以下操作：

   - 将 *YOUR_IDENTITY_POOL_ID* 替换为您在步骤 1：创建一个 Amazon Cognito 身份池和 IAM 角色中记下的 Amazon Cognito 身份池 ID。
   - 将区域值替换为为您的 Amazon S3 存储桶和 Amazon Cognito 身份池分配的区域。请注意，两种服务的区域必须相同（例如 us-east-2）。
   - 将 *bucket-name* 替换为您在步骤 3：添加 Amazon S3 存储桶和对象中创建的存储桶的名称。

替换完文本后，保存 `App.tsx` 文件。现在您可以运行该 Web 应用程序了。

步骤 5：运行示例

运行示例应用程序

1. 从命令行中导航到示例应用程序：

```
cd aws-doc-sdk-examples/javascriptv3/example_code/web/s3/list-objects/
```

2. 在命令行中，运行以下命令：

```
npm run dev
```

Vite 开发环境将运行，并显示以下消息：

```
VITE v4.3.9  ready in 280 ms

#  Local:   http://localhost:5173/
#  Network: use --host to expose
#  press h to show help
```

3. 在您的 Web 浏览器中，导航到上面显示的 URL（例如 http://localhost:5173）。该示例应用程序将向您显示 Amazon S3 存储桶中的对象文件名列表。

清理

要清除您在本教程中创建的资源，请执行以下操作：

- 在 Amazon S3 控制台中，删除创建的所有对象和所有存储桶（例如 getstartedbucket）。
- 在 IAM 控制台中，删除角色名称（例如 getStartedRole）。
- 在 Amazon Cognito 控制台中，删除身份池名称（例如 getStartedPool）。

将 SDK 设置为 JavaScript

本节中的主题说明了如何安装和加载的软件开发工具包， JavaScript 以便您可以访问该软件开发工具包支持的 Web 服务。

React Native 开发者应该使用 Amazon Amplify 它在上创建新项目 Amazon。详情请参阅aws-sdk-react-native档案。

主题

- 先决条件
- 安装适用于 JavaScript
- 加载适用的 SDK JavaScript

先决条件

在服务器上安装 Node.js（如果尚未安装）。

主题

- 设置 Amazon Node.js 环境
- 支持的 Web 浏览器

设置 Amazon Node.js 环境

要设置可以在其中运行应用程序 Amazon 的 Node.js 环境，请使用以下任一方法：

- 选择已预安装 Node.js 的 Amazon 机器映像 (AMI)。然后使用该 AMI 创建 Amazon EC2 实例。创建 Amazon EC2 实例时，从 Amazon Web Services Marketplace中选择您的 AMI。在中 Amazon Web Services Marketplace 搜索 Node.js，然后选择包含预装版本的 Node.js（32 位或 64 位）的 AMI 选项。
- 创建 Amazon EC2 实例并在该实例上安装 Node.js。有关如何在 Amazon Linux 实例上安装 Node.js 的更多信息，请参阅在 Amazon EC2 实例上设置 Node.js。

- 使用 Amazon Lambda 创建无服务器环境，将 Node.js 作为 Lambda 函数运行。有关在 Lambda 函数中使用 Node.js 的更多信息，请参阅《Amazon Lambda 开发人员指南》 中的编程模型 (Node.js)。
- 将你的 Node.js 应用程序部署到 Amazon Elastic Beanstalk。有关将 Node.js 与 Elastic Beanstalk 结合使用的更多信息，请参阅《Amazon Elastic Beanstalk 开发人员指南》 中的将 Node.js 应用程序部署到 Amazon Elastic Beanstalk。
- 使用创建 Node.js 应用服务器 Amazon OpsWorks。有关将 Node.js 与配合使用的更多信息 Amazon OpsWorks，请参阅《Amazon OpsWorks 用户指南》中的创建第一个 Node.js 堆栈。

支持的 Web 浏览器

Amazon SDK for JavaScript 支持所有现代 Web 浏览器。

在 3.183.0 或更高版本中，适用的 SDK JavaScript 使用 ES2020 构件，该构件支持以下最低版本。

浏览器	版本
Google Chrome	80.0+
Mozilla Firefox	80.0+
Opera	63.0+
Microsoft Edge	80.0+
Apple Safari	14.1+
Samsung Internet	12.0+

在 3.182.0 或更早版本中，适用的 SDK JavaScript 使用 ES5 工件，它支持以下最低版本。

浏览器	版本
Google Chrome	49.0+
Mozilla Firefox	45.0+
Opera	36.0+

浏览器	版本
Microsoft Edge	12.0+
Windows Internet Explorer	不适用
Apple Safari	9.0+
Android 浏览器	76.0+
UC 浏览器	12.12+
Samsung Internet	5.0+

> **Note**
>
> 诸如之类的框架 Amazon Amplify 可能无法提供与 SDK 相同的浏览器支持 JavaScript。有关详细信息，请参阅 Amazon Amplify 文档。

安装适用于 JavaScript

并非所有服务都可立即在 SDK 中或在所有 Amazon 地区提供。

要 Amazon SDK for JavaScript 通过使用 npm（Node.js 软件包管理器）安装服务，请在命令提示符下输入以下命令，其中 SERVICE 是###名称，例如s3。

```
npm install @aws-sdk/client-SERVICE
```

有关 Amazon SDK for JavaScript 服务客户端软件包的完整列表，请参阅 Amazon SDK for JavaScript API 参考指南。

加载适用的 SDK JavaScript

安装 SDK 之后，您可以使用 `import`，将客户端程序包加载到节点应用程序中。例如，要加载 Amazon S3 客户端和 Amazon S3 ListBuckets命令，请使用以下命令。

```
import { S3Client, ListBucketsCommand } from "@aws-sdk/client-s3";
```

将 SDK 配置为 JavaScript

在使用的 SDK 通过 API 调用 Web 服务之前，必须配置软件开发工具包。 JavaScript 至少，您必须配置以下项：

- 您将在哪个 Amazon 地区申请服务
- 您的代码如何进行身份验证 Amazon

除了这些设置，您可能还必须配置您的 Amazon 资源的权限。例如，您可以限制对 Amazon S3 存储桶的访问或者限制 Amazon DynamoDB 表只能进行只读访问。

S Amazon DK 和工具参考指南还包含许多软件开发工具包中常见的设置、功能和其他基础概念。 Amazon

本节中的主题介绍如何为 Node.js 配置软件开发工具包并在 Web 浏览器中 JavaScript 运行。 JavaScript

主题

- 每个服务的配置
- 设置 Amazon 区域
- 设置凭据
- Node.js 注意事项
- 浏览器脚本注意事项

每个服务的配置

您可以通过将配置信息传递给服务对象来配置 SDK。

服务级别配置提供了对各项服务的有效控制，使您能够在需求与默认配置不同时更新各项服务对象的配置。

> **Note**
>
> 在 2.x 版本中，可以将 Amazon SDK for JavaScript 服务配置传递给各个客户端构造函数。但是，这些配置将首先自动合并到全局 SDK 配置 `AWS.config` 的副本中。

此外，调用 `AWS.config.update({/* params *})` 仅更新在执行更新调用后实例化的服务客户端的配置，而不是任何现有客户端的配置。
这种行为经常引起混乱，因此很难向仅以向前兼容的方式影响一部分服务客户端的全局对象添加配置。在版本 3 中，不再有由 SDK 管理的全局配置。必须将配置传递至每个实例化的服务客户端。仍然可以在多个客户端之间共享相同的配置，但是该配置不会自动与全局状态合并。

为每项服务设置配置

您在 SDK 中使用的每项服务均通过服务对象进行访问，该服务对象是该服务 API 的一部分。JavaScript 例如，要访问亚马逊 S3 服务，您需要创建 Amazon S3 服务对象。您可以将特定于某项服务的配置设置指定为该服务对象的构造函数的一部分。

例如，如果您需要访问多个 Amazon 区域中的 Amazon EC2 对象，请为每个区域创建一个 Amazon EC2 服务对象，然后相应地设置每个服务对象的区域配置。

```
var ec2_regionA = new EC2({region: 'ap-southeast-2', maxAttempts: 15});
var ec2_regionB = new EC2({region: 'us-west-2', maxAttempts: 15});
```

设置 Amazon 区域

Amazon 区域是同一地理区域内的一组命名 Amazon 资源。区域的一个例子是 `us-east-1`，即美国东部（弗吉尼亚州北部）区域。在为的 SDK 中创建服务客户端时，您需要指定一个区域， JavaScript 这样 SDK 就可以访问该区域中的服务。有些服务仅在特定区域中提供。

默认情况下，的 SDK JavaScript 不选择区域。但是，您可以使用环境变量或共享配置`config`文件来设置 Amazon 区域。

在客户端类构造函数中

实例化服务对象时，可以将该资源的 Amazon 区域指定为客户端类构造函数的一部分，如下所示。

```
const s3Client = new S3.S3Client({region: 'us-west-2'});
```

使用环境变量

您可以使用 `AWS_REGION` 环境变量设置区域。如果您定义了此变量，则的 SDK 会 JavaScript 读取并使用它。

使用共享配置文件

就像共享凭据文件允许您存储证书以供 SDK 使用一样，您可以将 Amazon 区域和其他配置设置保存在名为 `config` SDK 使用的共享文件中。如果将`AWS_SDK_LOAD_CONFIG`环境变量设置为真实值，则 SDK 会在加载`config`文件时 JavaScript 自动搜索文件。保存 `config` 文件的位置取决于您的操作系统：

- Linux、macOS 或 Unix 用户 - `~/.aws/config`
- Windows 用户 - `C:\Users\USER_NAME\.aws\config`

如果您还没有共享 `config` 文件，您可以在指定的目录中创建一个。在以下示例中，`config` 文件设置区域和输出格式。

```
[default]
   region=us-west-2
   output=json
```

有关使用共享 `config` 和 `credentials` 文件的更多信息，请参阅 Amazon SDK 和工具参考指南 中的共享配置和凭证文件。

设置区域的优先顺序

区域设置的优先顺序如下：

1. 如果将某个区域传递给客户端类构造函数，则使用该区域。
2. 如果在环境变量中设置了某区域，则使用该区域。
3. 如果 `AMAZON_REGION` 环境变量是真值，则使用该区域。
4. 否则，将使用共享配置文件中定义的区域。

设置凭据

Amazon 使用证书来识别谁在呼叫服务以及是否允许访问所请求的资源。

无论是在 Web 浏览器中还是在 Node.js 服务器中运行，您的 JavaScript 代码都必须先获得有效的凭据，然后才能通过 API 访问服务。可以针对每个服务设置凭证，方法是将凭证直接传递给服务对象。

有几种方法可以设置凭证，这些方法在 Node.js 和 Web 浏览器 JavaScript中有所不同。本部分中的主题介绍如何在 Node.js 或 Web 浏览器中设置凭证。在每种情况下，选项以推荐顺序显示。

凭证的最佳实践

正确设置凭证可确保您的应用程序或浏览器脚本可以访问所需的服务和资源，同时最大限度地减少可能影响关键任务型应用程序或危及敏感数据的安全问题。

设置凭证时应用的一个重要原则是始终授予您的任务所需的最小权限。提供对资源的最小权限并根据需要添加更多权限更安全，而不是提供超过最小权限的权限，因此需要修复以后可能发现的安全问题。例如，除非您需要读取和写入单独的资源（例如 Amazon S3 存储桶或 DynamoDB 表中的对象），否则请将这些权限设置为只读。

有关授予最低权限的更多信息，请参阅《IAM 用户指南》最佳实践主题中的授予最低权限部分。

主题

- 在 Node.js 中设置凭据
- 在 Web 浏览器中设置凭据

在 Node.js 中设置凭据

我们建议在本地开发且雇主未向其提供身份验证方法的新用户进行设置 Amazon IAM Identity Center。有关更多信息，请参阅使用 Amazon 进行 SDK 身份验证。

Node.js 有几种方法可以为 SDK 提供凭证。其中一些方法更安全，而另一些方法则在开发应用程序时可以提供更大的便利。在 Node.js 中获取凭证时，请注意依赖多个源，例如环境变量和您加载的 JSON 文件。您可以更改运行代码的权限，而不会意识到已发生更改。

Amazon SDK for JavaScript V3 在 Node.js 中提供了默认的凭证提供者链，因此您无需明确提供凭证提供商。默认凭证提供程序链会尝试按给定优先级解析来自各种不同源的凭证，直到从其中一个源返回凭证。您可以在此处找到适用于 JavaScript V3 的 SDK 的凭证提供商链。

凭证提供程序链

所有 SDK 都有一系列地点（或源）供他们检查，以获取用于向 Amazon Web Service发出请求的有效凭证。找到有效凭证后，搜索即告停止。这种系统性搜索被称为默认凭证提供程序链。

对于链中的每个步骤，都有不同的设置值的方法。直接在代码中设置值始终优先，然后设置为环境变量，然后在共享 Amazon `config`文件中设置。有关更多信息，请参阅《Amazon SDK 和工具参考指南》中的设置的优先顺序。

《Amazon 软件开发工具包和工具参考指南》包含有关所有 SDK 使用的 S Amazon DK 配置设置的信息，以及。 Amazon CLI要详细了解如何通过共享 Amazon config文件配置 SDK，请参阅共享配置和凭据文件。要详细了解如何通过设置环境变量来配置 SDK，请参阅环境变量支持。

要进行身份验证 Amazon，请按下表所列的顺序 Amazon SDK for JavaScript 检查凭证提供商。

Amazon SDK for JavaScript 按优先级划@@ 分的 API 参考凭证提供者方法	可用的凭证提供程序	Amazon SDK 和工具参考指南
`fromEnv()`	Amazon 来自环境变量的访问密钥	Amazon 访问密钥
`fromSSO()`	Amazon IAM Identity Center。在本指南中，请参阅使用 Amazon 进行 SDK 身份验证。	IAM Identity Center 凭证提供程序
`fromIni()`	Amazon 来自共享`credentials` 文件`config`和文件的访问密钥	Amazon 访问密钥
	可信实体提供商（例如 `AWS_ROLE_ARN` ）	代入 IAM 角色
	来自 Amazon Security Token Service (Amazon STS) 的 Web 身份令牌	使用 Web 身份或 OpenID Connect 进行联合
	Amazon Elastic Container Service（Amazon ECS）凭证	容器凭证提供程序
	Amazon Elastic Compute Cloud (Amazon EC2) 实例配置文件凭证（IMDS 凭证提供程序）	IMDS 凭证提供程序
	流程凭证提供程序	流程凭证提供程序

Amazon SDK for JavaScript 按优先级划@@ 分的 API 参考凭证提供者方法	可用的凭证提供程序	Amazon SDK 和工具参考指南
	Amazon IAM Identity Center 证书	IAM Identity Center 凭证提供程序
fromProcess()	流程凭证提供程序	流程凭证提供程序
fromTokenFile()	来自 Amazon Security Token Service (Amazon STS) 的 Web 身份令牌	使用 Web 身份或 OpenID Connect 进行联合
fromContainerMetadata()	Amazon Elastic Container Service（Amazon ECS）凭证	容器凭证提供程序
fromInstanceMetadata()	Amazon Elastic Compute Cloud (Amazon EC2) 实例配置文件凭证（IMDS 凭证提供程序）	IMDS 凭证提供程序

如果您遵循推荐的新用户入门方法，则可以在入门主题的 使用 Amazon 进行 SDK 身份验证 中设置 Amazon IAM Identity Center 身份验证。其他身份验证方法适用于不同的情况。为避免安全风险，我们建议始终使用短期凭证。有关其他身份验证方法的过程，请参阅《Amazon SDK 和工具参考指南》 中的身份验证和访问。

本部分中的主题介绍如何将凭证加载到 Node.js 中。

主题

- 从 Amazon EC2 的 IAM 角色在 Node.js 中加载证书
- 加载 Node.js Lambda 函数的证书

从 Amazon EC2 的 IAM 角色在 Node.js 中加载证书

如果在 Amazon EC2 实例上运行 Node.js 应用程序，则可以利用 Amazon EC2 的 IAM 角色自动为实例提供凭证。如果将实例配置为使用 IAM 角色，则 SDK 会自动为您的应用程序选择 IAM 凭证，从而无需手动提供凭证。

有关将 IAM 角色添加到 Amazon EC2 实例的更多信息，请参阅适用于 Amazon EC2 的 IAM 角色。

加载 Node.js Lambda 函数的证书

创建 Amazon Lambda 函数时，必须创建一个有权执行该函数的特殊 IAM 角色。此角色称为执行角色。当您设置 Lambda 函数时，您必须指定您创建的 IAM 角色作为相应的执行角色。

执行角色为 Lambda 函数提供运行和调用其他 Web 服务所需的凭证。因此，您不需要为在 Lambda 函数中编写的 Node.js 代码提供凭证。

有关您创建 Lambda 执行角色的更多信息，请参阅《Amazon Lambda 开发人员指南》 中的管理权限：使用 IAM 角色（执行角色）。

在 Web 浏览器中设置凭据

有几种方法可以从浏览器脚本为 SDK 提供凭证。其中一些方法更安全，而另一些方法则在开发脚本时可以提供更大的便利。

下面是按推荐顺序提供凭证的方法：

1. 使用 Amazon Cognito 验证用户身份和提供凭证
2. 使用 Web 联合身份验证

> **Warning**
>
> 我们不建议在脚本中对您的 Amazon 凭据进行硬编码。硬编码凭证存在暴露您的访问密钥 ID 和秘密访问密钥的风险。

主题

• 使用 Amazon Cognito 身份验证用户身份

使用 Amazon Cognito 身份验证用户身份

获取浏览器脚本 Amazon 凭证的推荐方法是使用 Amazon Cognito 身份凭证客户端。`CognitoIdentityClient`Amazon Cognito 支持通过第三方身份提供商对用户进行身份验证。

要使用 Amazon Cognito Identity，您必须先在 Amazon Cognito 控制台中创建一个身份池。身份池表示应用程序为用户提供的身份组。为用户提供的身份唯一地标识每个用户账户。Amazon Cognito 身份

并不是凭证。使用 Amazon Security Token Service (Amazon STS) 中的 Web 联合身份验证支持将它们交换为凭证。

Amazon Cognito 可帮助您管理跨多个身份提供商的身份抽象。然后，在 Amazon STS中为凭证交换加载的身份。

配置 Amazon Cognito 身份凭证对象

如果您尚未创建身份池，则在配置 Amazon Cognito 客户端之前，请先创建一个以与 Amazon Cognito 控制台中的浏览器脚本一起使用。为身份池创建并关联经过身份验证和未经身份验证的 IAM 角色。有关更多信息，请参阅《Amazon Cognito 开发人员指南》中的教程：创建身份池。

未经身份验证的用户的身份未经过验证，因此，该角色很适合您的应用程序的来宾用户或用户身份验证与否无关紧要的情形。经过身份验证的用户可以通过证实其身份的第三方身份提供商登录到您的应用程序。确保您的资源的权限范围适当，让未经身份验证的用户无权访问这些资源。

配置身份池后，可使用 `@aws-sdk/credential-providers` 中的 `fromCognitoIdentityPool` 方法从身份池中检索凭证。在以下创建 Amazon S3 客户端的示例中，将 *AWS_REGION* 替换为区域，将 *IDENTITY_POOL_ID* 替换为身份池 ID。

```
// Import required AWS SDK clients and command for Node.js
import {S3Client} from "@aws-sdk/client-s3";
import {fromCognitoIdentityPool} from "@aws-sdk/credential-providers";

const REGION = AWS_REGION;

const s3Client = new S3Client({
  region: REGION,
  credentials: fromCognitoIdentityPool({
    clientConfig: { region: REGION }, // Configure the underlying
 CognitoIdentityClient.
    identityPoolId: 'IDENTITY_POOL_ID',
    logins: {
            // Optional tokens, used for authenticated login.
        },
  })
});
```

可选的 `logins` 属性是身份提供商名称到这些提供商身份令牌的映射。您如何从身份提供商获得令牌的方式取决于您使用的提供商。例如，如果您使用 Amazon Cognito 用户池作为身份验证提供商，则可以使用类似于以下方法的方法。

```
// Get the Amazon Cognito ID token for the user. 'getToken()' below.
let idToken = getToken();
let COGNITO_ID = "COGNITO_ID"; // 'COGNITO_ID' has the format 'cognito-
idp.REGION.amazonaws.com/COGNITO_USER_POOL_ID'
let loginData = {
  [COGNITO_ID]: idToken,
};
const s3Client = new S3Client({
    region: REGION,
    credentials: fromCognitoIdentityPool({
    clientConfig: { region: REGION }, // Configure the underlying
 CognitoIdentityClient.
    identityPoolId: 'IDENTITY_POOL_ID',
    logins: loginData
  })
});

// Strips the token ID from the URL after authentication.
window.getToken = function () {
  var idtoken = window.location.href;
  var idtoken1 = idtoken.split("=")[1];
  var idtoken2 = idtoken1.split("&")[0];
  var idtoken3 = idtoken2.split("&")[0];
  return idtoken3;
};
```

将未经身份验证的用户切换为经过身份验证的用户

Amazon Cognito 同时支持经过身份验证的用户和未经身份验证的用户。即使未经身份验证的用户不通过任何身份提供商登录，这些用户也有权访问您的资源。此级别的访问可用于向尚未登录的用户显示内容。即使每个未经身份验证的用户尚未单独登录和经过身份验证，这些用户在 Amazon Cognito 中也都具有唯一的身份。

最初未经身份验证的用户

用户通常从未经身份验证的角色开始，为此需要设置配置对象的凭证属性而不是 `logins` 属性。在这种情况下，您的默认凭证可能如下所示：

```
// Import the required Amazon SDK for JavaScript v3 modules.
import {fromCognitoIdentityPool} from "@aws-sdk/credential-providers";
// Set the default credentials.
```

```
const creds = new fromCognitoIdentityPool({
  IdentityPoolId: "IDENTITY_POOL_ID",
  clientConfig({ region: REGION }) // Configure the underlying CognitoIdentityClient.
});
```

切换为经过身份验证的用户

当未经身份验证的用户登录身份提供商并且您拥有令牌时，您可以通过调用可更新凭证对象和添加 `logins` 令牌的自定义函数，来将用户从未经身份验证的用户切换为经过身份验证的用户。

```
// Called when an identity provider has a token for a logged in user
function userLoggedIn(providerName, token) {
  creds.params.Logins = creds.params.logins || {};
  creds.params.Logins[providerName] = token;

  // Expire credentials to refresh them on the next request
  creds.expired = true;
}
```

Node.js 注意事项

尽管 Node.js 代码是 JavaScript，但 Amazon SDK for JavaScript 在 Node.js 中使用可能与在浏览器脚本中使用 SDK 有所不同。一些 API 方法在 Node.js 中有效，但在浏览器脚本以及其他方法中不起作用。成功使用某些 API 取决于您对常见 Node.js 代码编写模式的熟悉程度，例如导入和使用其他 Node.js 模块，如 `File System (fs)` 模块。

使用内置的 Node.js 模块

Node.js 提供了一组内置模块，无需安装即可使用它们。要使用这些模块，请使用 `require` 方法创建一个对象以指定模块名称。例如，要包含内置的 HTTP 模块，请使用以下方法。

```
import http from 'http';
```

调用模块的方法，就好像它们是该对象的方法一样。例如，下面的代码读取您的 HTML 文件。

```
// include File System module
import fs from "fs";
// Invoke readFile method
fs.readFile('index.html', function(err, data) {
  if (err) {
    throw err;
```

```
  } else {
    // Successful file read
  }
});
```

有关 Node.js 提供的所有内置模块的完整列表，请参阅 Node.js 网站上的 Node.js 文档。

使用 npm 软件包

除了内置模块，您还可以包含并合并来自 npm（即 Node.js 程序包管理器）的第三方代码。这是一个开源 Node.js 程序包的存储库和一个用于安装这些程序包的命令行界面。有关 npm 和当前可用程序包列表的更多信息，请参阅 https://www.npmjs.com。您还可以在此处了解可以使用的其他 Node.js 软件包 GitHub。

在 Node.js 中配置 maxSockets

在 Node.js 中，您可以设置每个源的最大连接数。如果设置了 `maxSockets`，则低级 HTTP 客户端会将请求排队，并在它们可用时将它们分配给套接字。

这使您可以设置在某个时间对给定源的并发请求数的上限。降低此值可以减少收到的限制或超时错误的数量。但是，它还会增加内存使用量，因为请求进行排队，直到套接字变为可用状态。

以下示例演示了如何为 DynamoDB 客户端设置 `maxSockets`：

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { NodeHttpHandler } from "@smithy/node-http-handler";
import https from "https";
let agent = new https.Agent({
  maxSockets: 25
});

let dynamodbClient = new DynamoDBClient({
  requestHandler: new NodeHttpHandler({
    requestTimeout: 3_000,
    httpsAgent: agent
  });
});
```

如果您不提供`maxSockets`值或`Agent`对象，则适用的 SDK 将 JavaScript 使用值 50。如果您提供一个`Agent`对象，则将使用其`maxSockets`值。有关`maxSockets`在 Node.js 中设置的更多信息，请参阅 Node.js 文档。

从 v3.521.0 起 Amazon SDK for JavaScript，您可以使用以下速记语法进行配置。requestHandler

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({
  requestHandler: {
    requestTimeout: 3_000,
    httpsAgent: { maxSockets: 25 },
  },
});
```

在 Node.js 中使用保持活动状态重用连接

默认的 Node.js HTTP/HTTPS 代理会为每个新请求创建一个新的 TCP 连接。为了避免建立新连接的成本，适用于 SDK 的 JavaScript 重复使用 TCP 连接。

对于短期操作（如 Amazon DynamoDB 查询），设置 TCP 连接的延迟开销可能大于操作本身。此外，由于 Dy namoDB 静态加密Amazon KMS与集成，因此您可能会遇到数据库延迟，必须为每个操作重新建立 Amazon KMS 新的缓存条目。

您也可以禁用在每个服务客户端上保持这些连接的活动状态，如以下 DynamoDB 客户端示例所示。

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { NodeHttpHandler } from "@smithy/node-http-handler";
import { Agent } from "http";
const dynamodbClient = new DynamoDBClient({
    requestHandler: new NodeHttpHandler({
        httpAgent: new Agent({ keepAlive: false })
    })
});
```

如果已启用 keepAlive，您还可以使用 keepAliveMsecs 设置 TCP Keep-Alive 数据包的初始延迟，默认值为 1000ms。有关详细信息，请参阅 Node.js 文档。

为 Node.js 配置代理

如果您无法直接连接到互联网，则适用的 SDK JavaScript 支持通过第三方 HTTP 代理使用 HTTP 或 HTTPS 代理。

要查找第三方 HTTP 代理，请在 npm 上搜索“HTTP 代理”。

要安装第三方 HTTP 代理的代理，请在命令提示符下输入以下内容，其中 *PROXY* 是 npm 软件包的名称。

```
npm install PROXY --save
```

要在应用程序中使用代理，请使用 `httpAgent` 和 `httpsAgent` 属性，如以下 DynamoDB 客户端示例所示。

```
import { DynamoDBClient } from '@aws-sdk/client-dynamodb';
import { NodeHttpHandler } from "@smithy/node-http-handler";
import { HttpsProxyAgent } from "hpagent";
const agent = new HttpsProxyAgent({ proxy: "http://internal.proxy.com" });
const dynamodbClient = new DynamoDBClient({
    requestHandler: new NodeHttpHandler({
        httpAgent: agent,
        httpsAgent: agent
    }),
});
```

> **Note**
>
> `httpAgent` 与 `httpsAgent`，而且由于来自客户端的大多数调用都是指向 `https`，因此两者都应设置。

在 Node.js 中注册证书包

Node.js 的默认信任存储包含访问 Amazon 服务所需的证书。在某些情况下，最好只包括一组特定的证书。

在本示例中，使用磁盘上的特定证书创建 `https.Agent`，除非提供指定的证书，否则它会拒绝连接。然后，DynamoDB 客户端将使用新创建的 `https.Agent`。

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { NodeHttpHandler } from "@smithy/node-http-handler";
import { Agent } from "https";
import { readFileSync } from "fs";
const certs = [readFileSync("/path/to/cert.pem")];
const agent = new Agent({
  rejectUnauthorized: true,
  ca: certs
```

```
});
const dynamodbClient = new DynamoDBClient({
  requestHandler: new NodeHttpHandler({
    httpAgent: agent,
    httpsAgent: agent
  })
});
```

浏览器脚本注意事项

以下主题描述了 Amazon SDK for JavaScript 在浏览器中使用脚本的特殊注意事项。

主题

- 为浏览器构建 SDK
- 跨源资源共享 (CORS)
- 将应用程序与 webpack 捆绑在一起

为浏览器构建 SDK

与 JavaScript 版本 2 (V2) 的 SDK 不同，V3 不是作为包含默认服务集支持的 JavaScript 文件提供的。取而代之的是，V3 允许您仅在浏览器中捆绑和包含所需 JavaScript 文件的 SDK，从而减少开销。我们建议使用 Webpack 将所需的 JavaScript 文件 SDK 以及您需要的任何其他第三方软件包捆绑到一个`Javascript`文件中，然后使用`<script>`标签将其加载到浏览器脚本中。有关 Webpack 的更多信息，请参阅将应用程序与 webpack 捆绑在一起。有关使用 Webpack 将适用的 V3 SDK 加载 JavaScript 到浏览器的示例，请参阅构建应用程序以将数据提交到 DynamoDB。

如果您在浏览器中强制执行 CORS 的环境之外使用 SDK，并且想要访问 SDK 为其提供的所有服务 JavaScript，则可以通过克隆存储库并运行与构建 SDK 的默认托管版本相同的构建工具在本地构建 SDK 的自定义副本。下面几部分介绍使用额外服务和 API 版本构建 SDK 的步骤。

使用 SDK 生成器构建 SDK JavaScript

> **Note**
>
> Amazon Web Services 版本 3 (V3) 不再支持浏览器生成器。为了最大限度地减少浏览器应用程序的带宽使用量，我们建议您导入命名模块，然后捆绑它们以减小大小。有关捆绑的更多信息，请参阅将应用程序与 webpack 捆绑在一起。

跨源资源共享 (CORS)

跨源资源共享（即 CORS）是一项现代 Web 浏览器的安全功能。它使得 Web 浏览器可以协商哪些域能够发出对外部网站或服务的请求。

在使用 Amazon SDK for JavaScript 开发浏览器应用程序时，CORS 是一个重要的考虑因素，因为对资源的大部分请求发送到外部域，例如 Web 服务的端点。如果您的 JavaScript 环境强制执行 CORS 安全，则必须使用该服务配置 CORS。

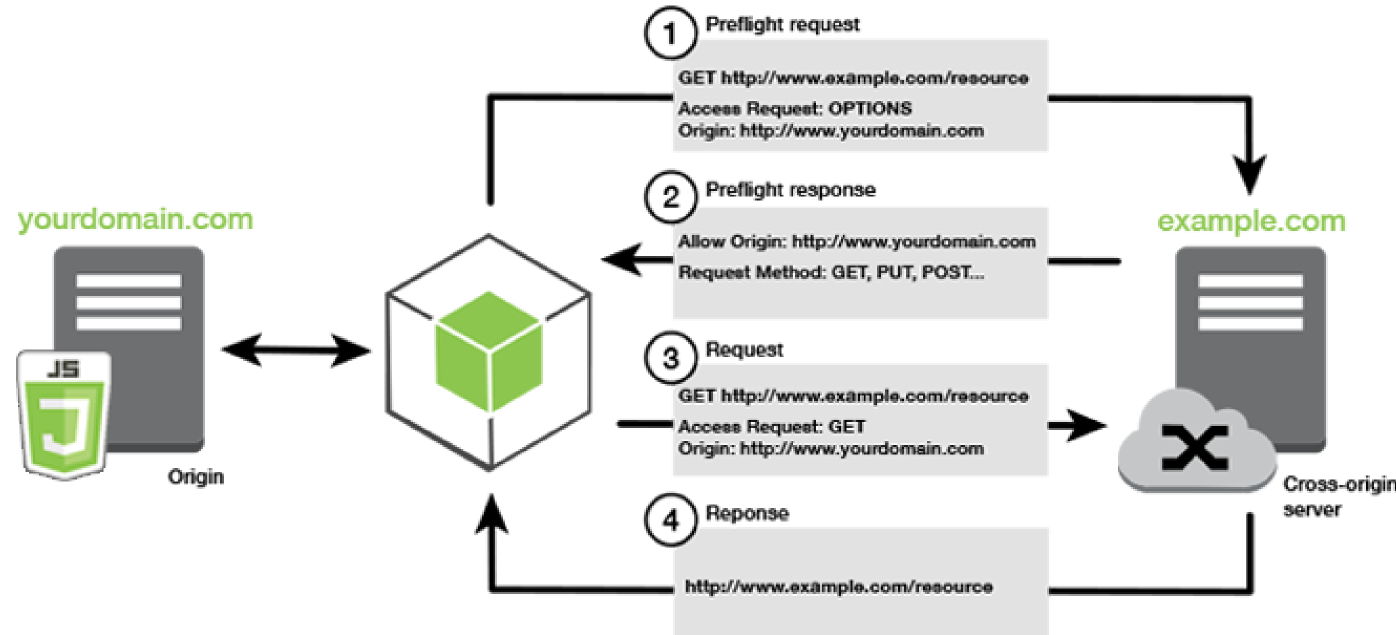
CORS 根据以下条件，确定是否允许跨源请求中的共享：

- 发出请求的特定域
- 发出的 HTTP 请求的类型（GET、PUT、POST、DELETE 等等）

CORS 工作原理

在最简单的情况下，浏览器脚本从其他域中的服务器发出对某个资源的 GET 请求。根据该服务器的 CORS 配置，如果请求来自已授权提交 GET 请求的域，则跨来源服务器通过返回请求的资源做出响应。

如果请求域或者 HTTP 请求的类型未获得授权，则将拒绝请求。但是，CORS 实现了在实际提交请求之前进行预检。在这种情况下将提交预检请求，在其中发送 `OPTIONS` 访问请求操作。如果跨来源服务器的 CORS 配置授予对请求域的访问权限，则服务器发送回预检响应，其中列出请求域可以对所请求资源发出的所有 HTTP 请求类型。



是否需要 CORS 配置？

Amazon S3 桶需要 CORS 配置，然后才能在桶上执行操作。在某些 JavaScript 环境中，可能无法强制执行 CORS，因此没有必要配置 CORS。例如，如果您在 Amazon S3 桶中托管应用程序并访问 `*.s3.amazonaws.com` 或某个其它特定端点的资源，您的请求不会访问外部域。因此，此配置不需要 CORS。在这种情况下，Amazon S3 之外的服务仍使用 CORS。

为 Amazon S3 存储桶配置 CORS

您可以在 Amazon S3 控制台中配置 Amazon S3 桶，以使用 CORS。

如果您要在 Amazon Web 服务管理控制台中配置 CORS，则必须使用 JSON 来创建 CORS 配置。新的 Amazon Web 服务管理控制台仅支持 JSON CORS 配置。

> **Important**
>
> 在新的 Amazon Web 服务管理控制台中，CORS 配置必须为 JSON。

1. 在 Amazon Web 服务管理控制台中，打开 Amazon S3 控制台，找到要配置的存储桶，然后选中其复选框。
2. 在打开的窗格中，选择权限。
3. 在权限选项卡中，选择 CORS 配置。

4. 在 CORS 配置编辑器 中输入您的 CORS 配置，然后选择保存。

CORS 配置是一个 XML 文件，在 `<CORSRule>` 中包含了一系列规则。一个配置最多可以有 100 个规则。规则由以下标签之一定义：

- `<AllowedOrigin>` - 指定您允许发出跨域请求的域源。
- `<AllowedMethod>` - 指定您允许在跨域请求中使用的请求类型（GET、PUT、POST、DELETE、HEAD）。
- `<AllowedHeader>` - 指定预检请求中允许的标头。

有关示例配置，请参阅《Amazon Simple Storage Service 用户指南》中的[如何在我的存储桶上配置 CORS？](#)。

CORS 配置示例

以下 CORS 配置示例允许用户从域 `example.org` 中查看、添加、移除或更新存储桶内的对象。不过，我们建议您将 `<AllowedOrigin>` 的范围限定到您的网站域名。您可以指定 `"*"` 以允许任意源。

> **Important**
>
> 在新的 S3 控制台中，CORS 配置必须是 JSON。

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<CORSConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <CORSRule>
    <AllowedOrigin>https://example.org</AllowedOrigin>
    <AllowedMethod>HEAD</AllowedMethod>
    <AllowedMethod>GET</AllowedMethod>
    <AllowedMethod>PUT</AllowedMethod>
    <AllowedMethod>POST</AllowedMethod>
    <AllowedMethod>DELETE</AllowedMethod>
    <AllowedHeader>*</AllowedHeader>
    <ExposeHeader>ETag</ExposeHeader>
    <ExposeHeader>x-amz-meta-custom-header</ExposeHeader>
  </CORSRule>
</CORSConfiguration>
```

JSON

```
[
    {
        "AllowedHeaders": [
            "*"
        ],
        "AllowedMethods": [
            "HEAD",
            "GET",
            "PUT",
            "POST",
            "DELETE"
        ],
        "AllowedOrigins": [
            "https://www.example.org"
        ],
        "ExposeHeaders": [
             "ETag",
             "x-amz-meta-custom-header"]
    }
]
```

此配置不授权用户在存储桶上执行操作。它使浏览器的安全模型允许对 Amazon S3 的请求。必须通过存储桶权限或 IAM 角色权限来配置权限。

您可以使用 `ExposeHeader`，让 SDK 读取从 Amazon S3 返回的响应标头。例如，如果要从 PUT 或分段上传读取 `ETag` 标头，则需要在配置中包括 `ExposeHeader` 标签，如上例中所示。SDK 只能访问通过 CORS 配置公开的标头。如果您在对象上设置元数据，则将值作为标头返回并带有 `x-amz-meta-` 前缀，例如 `x-amz-meta-my-custom-header`，并且也必须通过相同的方式公开。

将应用程序与 webpack 捆绑在一起

浏览器脚本或 Node.js 中使用代码模块的 Web 应用程序会创建依赖关系。这些代码模块可能会具有自身的依赖关系，导致您的应用程序需要一组互连的模块才能正常工作。要管理依赖关系，您可以使用 `webpack` 等模块捆绑程序。

`webpack` 模块捆绑程序解析您的应用程序代码，搜索 `import` 或 `require` 语句，创建包含您应用程序所需的全部资产的捆绑。这样可以轻松地通过网页提供资产服务。的 SDK JavaScript 可以`webpack`作为依赖项之一包含在输出包中。

有关更多信息webpack，请参阅上的 webpack 模块捆绑器。 GitHub

安装 webpack

要安装 webpack 模块捆绑程序，您必须已经安装了 npm（Node.js 程序包管理器）。键入以下命令安装 webpack CLI 和 JavaScript 模块。

```
npm install --save-dev webpack
```

要使用 path 模块来处理文件和目录路径（该模块是通过 webpack 自动安装的），您可能需要安装 Node.js path-browserify 软件包。

```
npm install --save-dev path-browserify
```

配置 webpack

默认情况下，Webpack 会搜索项目根目录webpack.config.js中名为的 JavaScript 文件。此文件指定您的配置选项。以下是 5.0.0 及更高 WebPack 版本的webpack.config.js配置文件示例。

> **Note**
>
> Webpack 配置要求因您安装的 Webpack 版本而异。有关更多信息，请参阅 Webpack 文档。

```
// Import path for resolving file paths
var path = require("path");
module.exports = {
  // Specify the entry point for our app.
  entry: [path.join(__dirname, "browser.js")],
  // Specify the output file containing our bundled code.
  output: {
    path: __dirname,
    filename: 'bundle.js'
  },
   // Enable WebPack to use the 'path' package.
   resolve:{
  fallback: { path: require.resolve("path-browserify")}
  }
  /**
  * In Webpack version v2.0.0 and earlier, you must tell
  * webpack how to use "json-loader" to load 'json' files.
```

```
  * To do this Enter 'npm --save-dev install json-loader' at the
  * command line to install the "json-loader' package, and include the
  * following entry in your webpack.config.js.
  * module: {
    rules: [{test: /\.json$/, use: use: "json-loader"}]
  }
  **/
};
```

在本示例中，指定 `browser.js` 为入口点。入口点 是 `webpack` 开始搜索导入的模块所用的文件。输出的文件名指定为 `bundle.js`。此输出文件将包含应用程序运行所需的所有内容。 JavaScript 如果入口点中指定的代码导入或需要其他模块（例如的 SDK） JavaScript，则无需在配置中指定该代码即可捆绑该代码。

运行 webpack

要生成应用程序以使用 `webpack`，请将以下内容添加到您 `package.json` 文件的 `scripts` 对象。

```
"build": "webpack"
```

以下是演示如何添加 `webpack` 的示例 `package.json` 文件。

```
{
  "name": "aws-webpack",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "build": "webpack"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "@aws-sdk/client-iam": "^3.32.0",
    "@aws-sdk/client-s3": "^3.32.0"
  },
  "devDependencies": {
    "webpack": "^5.0.0"
  }
}
```

要生成应用程序，请输入以下命令。

```
npm run build
```

然后，webpack模块捆绑器会生成您在项目根目录中指定的 JavaScript 文件。

使用 webpack 捆绑包

要在浏览器脚本中使用捆绑，您可以使用 <script> 标签整合捆绑，如下例中所示。

```
<!DOCTYPE html>
<html>
    <head>
        <title>Amazon SDK with webpack</title>
    </head>
    <body>
        <div id="list"></div>
        <script src="bundle.js"></script>
    </body>
</html>
```

Node.js 的捆绑包

您可以通过在配置中将 node 指定为目标，使用 webpack 生成在 Node.js 中运行的捆绑。

```
target: "node"
```

在磁盘空间有限的环境中运行 Node.js 应用程序时，这非常有用。此处是将 Node.js 指定为输出目标的示例 webpack.config.js 配置。

```
// Import path for resolving file paths
var path = require("path");
module.exports = {
  // Specify the entry point for our app.
  entry: [path.join(__dirname, "browser.js")],
  // Specify the output file containing our bundled code.
  output: {
    path: __dirname,
    filename: 'bundle.js'
  },
  // Let webpack know to generate a Node.js bundle.
```

```
  target: "node",
   // Enable WebPack to use the 'path' package.
   resolve:{
  fallback: { path: require.resolve("path-browserify")}
   /**
   * In Webpack version v2.0.0 and earlier, you must tell
   * webpack how to use "json-loader" to load 'json' files.
   * To do this Enter 'npm --save-dev install json-loader' at the
   * command line to install the "json-loader' package, and include the
   * following entry in your webpack.config.js.
   module: {
    rules: [{test: /\.json$/, use: use: "json-loader"}]
  }
  **/
};
```

使用 SDK 中的 Amazon 服务 JavaScript

Amazon SDK for JavaScript v3 通过一系列客户端类提供对其支持的服务的访问权限。从这些客户端类，您可以创建服务接口对象，这些对象通常称为服务对象。每项支持的 Amazon 服务都有一个或多个客户端类，这些类提供用于使用服务功能和资源的低级 API。例如，Amazon DynamoDB API 通过 DynamoDB 类提供。

通过 SDK 公开的服务 JavaScript 遵循请求-响应模式与调用应用程序交换消息。在此模式中，调用服务的代码向服务的端点提交 HTTP/HTTPS 请求。请求中包含成功调用特定功能所需的参数。调用的服务将生成发送回请求方的响应。如果操作成功，则响应包含数据，如果操作不成功，则包含错误消息。

调用 Amazon 服务包括对服务对象执行操作的完整请求和响应生命周期，包括尝试的任何重试。一个请求包含零个或多个属性作为 JSON 参数。响应被封装在与操作相关的对象中，并通过几种技术（例如回调函数或承诺）之一返回给请求者。 JavaScript

主题

- 创建和调用服务对象
- 异步呼叫服务
- 创建服务客户端请求
- 处理服务客户端的响应
- 使用 JSON
- JavaScript 代码示例的 SDK

创建和调用服务对象

该 JavaScript API 支持大多数可用 Amazon 服务。 JavaScriptAPI 中的每项服务都为客户端类提供了一个send方法，您可以使用该方法来调用该服务支持的每个 API。有关 JavaScript API 中的服务类、操作和参数的更多信息，请参阅 API 参考。

在 Node.js 中使用 SDK 时，您使用 import 将每个所需服务的 SDK 添加到应用程序，这为所有当前服务提供支持。以下示例在 us-west-1 区域中创建一个 Amazon S3 服务对象。

```
// Import the Amazon S3 service client
import { S3Client } from "@aws-sdk/client-s3";
// Create an S3 client in the us-west-1 Region
const s3Client = new S3Client({
    region: "us-west-1"
```

```
});
```

指定服务对象参数

调用服务对象的方法时，根据 API 的需要在 JSON 中传递参数。例如，在 Amazon S3 中，要获取指定存储桶和密钥的数据元，请将以下参数传递给`GetObjectCommand`方法`S3Client`。有关传递 JSON 参数的更多信息，请参阅使用 JSON。

```
s3Client.send(new GetObjectCommand({Bucket: 'bucketName', Key: 'keyName'}));
```

有关亚马逊 S3 参数的更多信息，请参阅 API 参考中的 @aws-sdk/client-s3。

异步呼叫服务

通过 SDK 发出的所有请求均为异步。在编写浏览器脚本时，请务必记住这一点。 JavaScript 在 Web 浏览器中运行通常只有一个执行线程。对 Amazon 服务进行异步调用后，浏览器脚本继续运行，在此过程中，浏览器脚本可以尝试在返回之前执行依赖于该异步结果的代码。

对 Amazon 服务进行异步调用包括管理这些调用，这样您的代码就不会在数据可用之前尝试使用数据。本部分中的主题说明管理异步调用的需求，以及在管理它们时可以使用的具体不同技术。

尽管您可以使用这些技术中的任何一种来管理异步调用，但我们建议您对所有新代码使用异步/等待。

异步/等待

  我们建议您使用此技术，因为这是 V3 中的默认行为。

Promise

  在不支持异步/等待的浏览器中使用此技术。

回调

  除非在非常简单的情况下，否则请避免使用回调。但是，您可能会发现它对迁移场景很有用。

主题

- 管理异步调用
- 使用异步/等待
- 使用 JavaScript 承诺
- 使用匿名回调函数

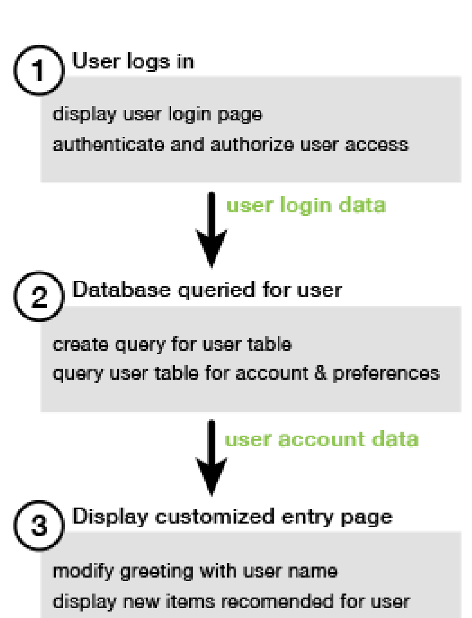
管理异步调用

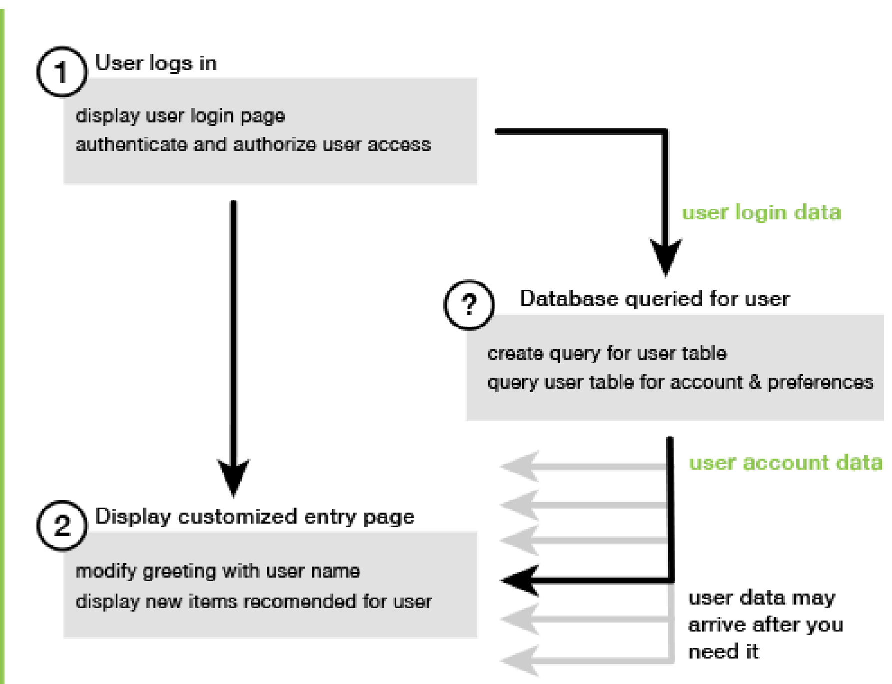
例如，电子商务网站的主页会让返回的客户登录。客户登录可以获得的一部分好处在于，登录之后网站可以根据其特定首选项进行自定义。要做到这一点：

1. 客户必须登录并使用其登录凭证进行验证。
2. 从客户数据库中请求客户的首选项。
3. 数据库提供客户的首选项，这些首选项用于在页面加载之前自定义网站。

如果这些任务同步执行，则必须在每个任务完成之后才能执行下一个任务。在数据库返回客户首选项之前，网页无法完成加载。但是，在数据库查询发送到服务器之后，由于网络瓶颈、极高的数据库流量或者糟糕的移动设备连接，客户数据的接收可能会延迟甚至失败。

要避免网站在这些情况下停滞不前，可以异步调用数据库。数据库调用执行之后，发送您的异步请求，您的代码继续按预期方式执行。如果您未能正确地管理异步调用的响应，代码会在数据尚不可用时，尝试使用预期从数据库返回的信息。





使用异步/等待

您应该考虑使用异步/等待，而不是 Promise。与使用 Promise 相比，异步函数更简单，并且需要的样板文件更少。等待只能在异步函数中用于异步等待值。

以下示例使用异步/等待来列出您在 us-west-2 中的所有 Amazon DynamoDB 表。

> **Note**
>
> 运行此示例需执行的操作：
>
> - 通过在项目的 Amazon SDK for JavaScript 命令行中npm install @aws-sdk/client-dynamodb输入来安装 DynamoDB 客户端。
> - 确保您的 Amazon 凭证配置正确。有关更多信息，请参阅设置凭据。

```
import { DynamoDBClient,
ListTablesCommand } from "@aws-sdk/client-dynamodb";
(async function () {
    const dbClient = new DynamoDBClient({ region: "us-west-2" });
    const command = new ListTablesCommand({});

    try {
        const results = await dbClient.send(command);
        console.log(results.TableNames.join('\n'));
    } catch (err) {
        console.error(err)
    }
})();
```

> **Note**
>
> 并非所有浏览器都支持异步/等待。有关支持异步/等待的浏览器列表，请参阅异步函数。

使用 JavaScript 承诺

使用服务客户端的 Amazon SDK for JavaScript v3 方法 (ListTablesCommand) 进行服务调用和管理异步流，而不是使用回调。以下示例演示如何获取 us-west-2 中您的 Amazon DynamoDB 表的名称。

```
import { DynamoDBClient,
        ListTablesCommand
} from "@aws-sdk/client-dynamodb";
const dbClient = new DynamoDBClient({ region: 'us-west-2' });
```

```
dbClient
  .listtables(new ListTablesCommand({}))
  .then(response => {
    console.log(response.TableNames.join('\n'));
  })
  .catch((error) => {
    console.error(error);
  });
```

协调多项承诺

在某些情况下，您的代码必须进行多个异步调用，这些调用只有在它们都成功返回时才需要执行操作。如果您使用 promise 管理这些单独的异步方法调用，则可以创建使用 `all` 方法的额外 promise。

此方法只有在执行了您传递到方法中的 promise 数组时，才会执行此伞形 promise。回调函数将 promise 的值数组传递到 `all` 方法。

在以下示例中，一个 Amazon Lambda 函数必须对 Amazon DynamoDB 进行三次异步调用，但只能在每个调用的承诺兑现后才能完成。

```
const values = await Promise.all([firstPromise, secondPromise, thirdPromise]);

console.log("Value 0 is " + values[0].toString);
console.log("Value 1 is " + values[1].toString);
console.log("Value 2 is " + values[2].toString);

return values;
```

Promise 的浏览器和 Node.js 支持

对原生 JavaScript 承诺（ECMAScript 2015）的支持取决于执行代码的 JavaScript引擎和版本。为了帮助确定您的代码需要运行的每个环境中对 JavaScript 承诺的支持，请参阅上的 ECMAScript 兼容性表。 GitHub

使用匿名回调函数

每个服务对象方法都可以接受匿名回调函数作为最后一个参数。此回调函数的签名如下。

```
function(error, data) {
    // callback handling code
};
```

此回调函数在返回成功响应或错误数据时执行。如果方法调用成功，则响应的内容在 data 参数中供回调函数使用。如果调用不成功，则在 error 参数中提供有关失败的详细信息。

通常，回调函数内部的代码经过了错误测试，在返回错误时会进行处理。如果未返回错误，则代码从 data 参数检索响应中的数据。回调函数的基本格式如此例中所示。

```
function(error, data) {
    if (error) {
        // error handling code
        console.log(error);
    } else {
        // data handling code
        console.log(data);
    }
};
```

在以上示例中，错误的详细信息或者返回的数据记录到控制台中。此处的示例演示了作为对服务对象调用方法的一部分传递的回调函数。

```
ec2.describeInstances(function(error, data) {
  if (error) {
    console.log(error); // an error occurred
  } else {
    console.log(data); // request succeeded
  }
});
```

创建服务客户端请求

向 Amazon 服务客户提出请求很简单。适用于 SDK 的版本 3 (V3) JavaScript 允许您发送请求。

> **Note**
>
> 使用适用于 SDK 的 V3 时，也可以使用版本 2 (V2) 命令执行操作。 JavaScript有关更多信息，请参阅使用 V2 命令。

发送请求：

1. 使用所需的配置初始化一个客户端对象，例如一个特定的 Amazon 区域。

2. （可选）使用请求的值（例如特定 Amazon S3 存储桶的名称）创建请求 JSON 对象。您可以检查请求的参数，方法是查看“API 参考”主题以了解具有与客户端方法关联的名称的接口。例如，如果您使用*AbcCommand*客户端方法，则请求接口为*AbcInput*。
3. （可选）使用请求对象作为输入来初始化服务命令。
4. 使用命令对象作为输入在客户端上调用 `send`。

例如，要列出您在 `us-west-2` 的 Amazon DynamoDB 表，可以使用异步/等待来完成。

```
import {
DynamoDBClient,
ListTablesCommand
} from "@aws-sdk/client-dynamodb";

(async function() {
const dbClient = new DynamoDBClient({ region: 'us-west-2' });
const command = new ListTablesCommand({});

try {
  const results = await dbClient.send(command);
  console.log(results.TableNames.join('\n'));
} catch (err) {
  console.error(err);
}
})();
```

处理服务客户端的响应

调用服务客户端方法后，它会返回一个接口的响应对象实例，其名称与该客户端方法相关联。例如，如果您使用 *AbcCommand*客户端方法，则响应对象的类型为 *AbcResponse*（接口）。

访问响应中返回的数据

响应对象包含服务请求返回的数据作为属性。

在创建服务客户端请求中， `ListTablesCommand` 命令在响应的 `TableNames` 属性中返回了表名。

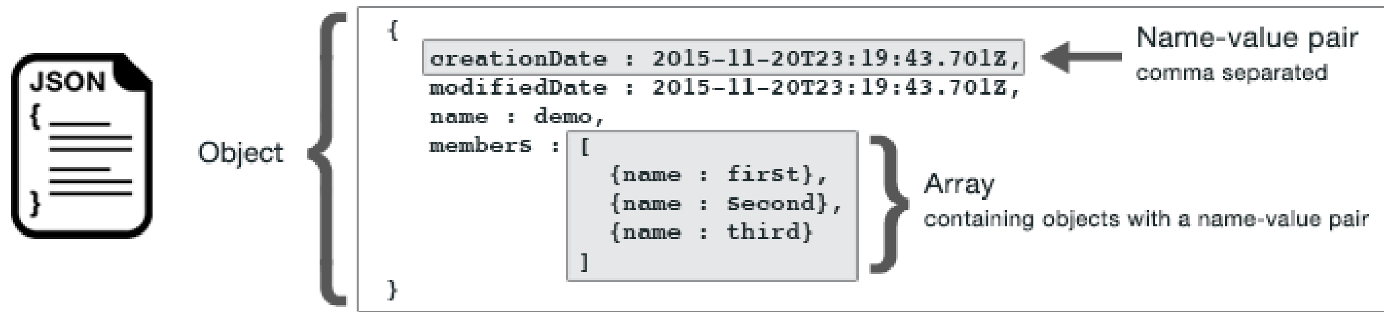
访问错误信息

如果命令失败，则将引发异常。您可以根据需要处理异常。

使用 JSON

JSON 是一种数据交换格式，便于人类阅读，并且是机器可读的。尽管 JSON 这个名字是JavaScript 对象表示法的缩写，但 JSON 的格式与任何编程语言无关。

在发出请求时 Amazon SDK for JavaScript 使用 JSON 向服务对象发送数据，并以 JSON 形式接收来自服务对象的数据。有关 JSON 的更多信息，请参阅 json.org。



JSON 通过两种方式表示数据：

- 对象，其是无序名称-值对集合。对象在左大括号 ({) 和右大括号 (}) 内定义。每个名称-值对以名称开头，后接一个冒号，再接值。名称/值对以逗号分隔。
- 数组，其是有序值集合。数组在左方括号 ([) 和右方括号 (]) 内定义。数组中的项目以逗号分隔。

下面是 JSON 对象示例，其中包含一个对象数组，这些对象表示扑克游戏中的扑克。每张扑克都由两个名称/值对定义，一个指定用于表示扑克的唯一值，另一个指定指向对应扑克图像的 URL。

```
var cards = [
  {"CardID":"defaultname", "Image":"defaulturl"},
  {"CardID":"defaultname", "Image":"defaulturl"},
  {"CardID":"defaultname", "Image":"defaulturl"},
  {"CardID":"defaultname", "Image":"defaulturl"},
  {"CardID":"defaultname", "Image":"defaulturl"}
];
```

将 JSON 作为服务对象参数

以下是一个简单 JSON 示例，用于定义对 Amazon Lambda 服务对象的调用的参数。

```
const params = {
```

```
    FunctionName : funcName,
    Payload : JSON.stringify(payload),
    LogType : LogType.Tail,
};
```

params 对象由三个名称/值对定义，在左右大括号中以逗号分隔。向服务对象方法调用提供参数时，名称由您计划调用的服务对象方法的参数名称确定。调用 Lambda 函数时，FunctionName、Payload 和 LogType 是用于在 Lambda 服务对象上调用 invoke 方法的参数。

将参数传递给服务对象方法调用时，将 JSON 对象提供给方法调用，如下面调用 Lambda 函数的示例中所示。

```
const invoke = async (funcName, payload) => {
  const client = new LambdaClient({});
  const command = new InvokeCommand({
    FunctionName: funcName,
    Payload: JSON.stringify(payload),
    LogType: LogType.Tail,
  });

  const { Payload, LogResult } = await client.send(command);
  const result = Buffer.from(Payload).toString();
  const logs = Buffer.from(LogResult, "base64").toString();
  return { logs, result };
};
```

JavaScript 代码示例的 SDK

本节中的主题包含如何与各种服务的 API Amazon SDK for JavaScript 一起使用来执行常见任务的示例。

在上的 "代码示例存储库" 中查找这些示例和其他示例的源Amazon 代码 GitHub。要提出一个新的代码示例供 Amazon 文档团队考虑制作，请创建请求。该团队正在寻求生成涵盖更多应用场景和使用情形的代码示例，而不仅仅是涵盖个别 API 调用的简单代码片段。有关说明，请参阅的贡献指南中的 "创作代码" 部分。 GitHub

> **Important**
>
> 这些示例使用 ECMAScript6 导入/导出语法。

- 这需要使用 Node.js 版本 14.17 或更高版本。要下载并安装最新版本的 Node.js，请参阅 Node.js 下载。
- 如果您更喜欢使用 CommonJS 语法，请参阅 JavaScript ES6/CommonJS 语法以获取转换指南。

主题

- JavaScript ES6/CommonJS 语法
- Amazon DynamoDB 示例
- AWS Elemental MediaConvert 示例
- Amazon Lambda 示例
- Amazon Lex 示例
- Amazon Polly 示例
- Amazon Redshift 示例
- Amazon Simple Email Service 示例
- Amazon Simple Notification Service 示例
- Amazon Transcribe 示例
- 在 Amazon EC2 实例上设置 Node.js
- 构建应用程序以将数据提交到 DynamoDB
- 针对经过身份验证的用户构建转录应用程序
- 使用 API Gateway 调用 Lambda
- 使用 Amazon SDK for JavaScript 创建 Amazon 无服务器工作流
- 创建计划事件以执行 Amazon Lambda 函数
- 构建 Amazon Lex 聊天机器人
- 创建示例消息收发应用程序

JavaScript ES6/CommonJS 语法

Amazon SDK for JavaScript 代码示例是使用 ECMAScript 6 (ES6) 编写的。ES6 带来了新的语法和新功能，使您的代码更现代、更具可读性，并能做到更多的事情。

要使用 ES6，您需要使用 Node.js 版本 13.x 或更高版本。要下载并安装最新版本的 Node.js，请参阅 Node.js 下载。但是，如果您愿意，可以使用以下指南将我们的任何示例转换为 CommonJS 语法：

- 从您的项目环境中的 `package.json` 中移除 `"type" : "module"`。
- 将所有 ES6 `import` 语句转换为 CommonJS `require` 语句。例如，将以下内容：

```
import { CreateBucketCommand } from "@aws-sdk/client-s3";
import { s3 } from "./libs/s3Client.js";
```

  转换为其 CommonJS 等效语句：

```
const { CreateBucketCommand } = require("@aws-sdk/client-s3");
const { s3 } = require("./libs/s3Client.js");
```

- 将所有 ES6 `export` 语句转换为 CommonJS `module.exports` 语句。例如，将以下内容：

```
export {s3}
```

  转换为其 CommonJS 等效语句：

```
module.exports = {s3}
```

以下示例演示了用于在 ES6 和 CommonJS 中创建 Amazon S3 存储桶的代码示例。

ES6

libs/s3Client.js

```
// Create service client module using ES6 syntax.
import { S3Client } from "@aws-sdk/client-s3";
// Set the AWS region
const REGION = "eu-west-1"; //e.g. "us-east-1"
// Create Amazon S3 service object.
const s3 = new S3Client({ region: REGION });
// Export 's3' constant.
export {s3};
```

s3_createbucket.js

```
// Get service clients module and commands using ES6 syntax.
 import { CreateBucketCommand } from "@aws-sdk/client-s3";
 import { s3 } from "./libs/s3Client.js";

// Get service clients module and commands using CommonJS syntax.
// const { CreateBucketCommand } = require("@aws-sdk/client-s3");
// const { s3 } = require("./libs/s3Client.js");

// Set the bucket parameters
const bucketParams = { Bucket: "BUCKET_NAME" };

// Create the Amazon S3 bucket.
const run = async () => {
  try {
    const data = await s3.send(new CreateBucketCommand(bucketParams));
    console.log("Success", data.Location);
    return data;
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

CommonJS

libs/s3Client.js

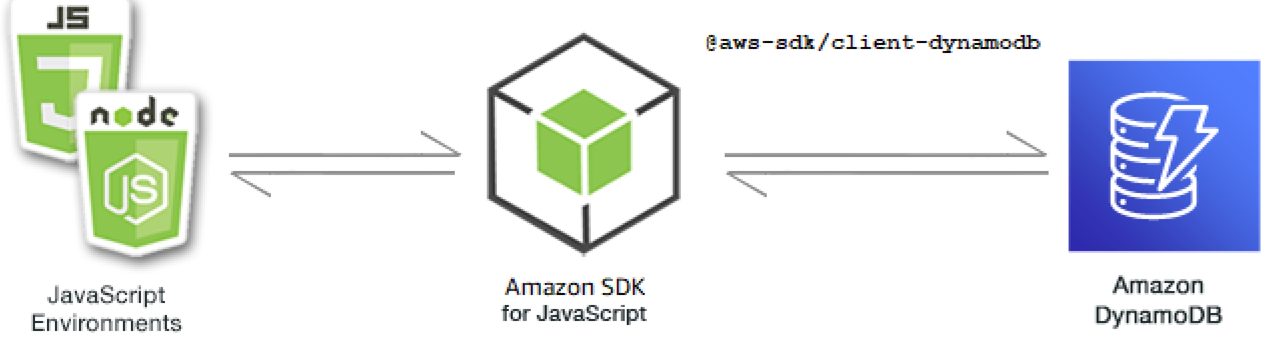
```
// Create service client module using CommonJS syntax.
 const { S3Client } = require("@aws-sdk/client-s3");
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
 // Create Amazon S3 service object.
const s3 = new S3Client({ region: REGION });
// Export 's3' constant.
 module.exports ={s3};
```

s3_createbucket.js

```
// Get service clients module and commands using CommonJS syntax.
const { CreateBucketCommand } = require("@aws-sdk/client-s3");
const { s3 } = require("./libs/s3Client.js");

// Set the bucket parameters
const bucketParams = { Bucket: "BUCKET_NAME" };

// Create the Amazon S3 bucket.
const run = async () => {
  try {
    const data = await s3.send(new CreateBucketCommand(bucketParams));
    console.log("Success", data.Location);
    return data;
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

Amazon DynamoDB 示例

Amazon DynamoDB 是一种完全托管的 NoSQL 云数据库，支持文档和键值两种存储模式。您可以为数据创建无架构表，而无需预配置或维护专用的数据库服务器。



适用于 DynamoDB 的 JavaScript API 通过 `DynamoDB`、`DynamoDBStreams` 和 `DynamoDB.DocumentClient` 客户端类公开。有关使用 DynamoDB 客户端类的更多信息，请参阅《API 参考》中的类：DynamoDB、类：AWS.DynamoDBStreams 和类：DynamoDB 实用工具。

主题

- 在 DynamoDB 中创建和使用表
- 在 DynamoDB 中读取和写入单个项目
- 在 DynamoDB 中批量读取和写入项目
- 查询并扫描 DynamoDB 表
- 使用 DynamoDB 文档客户端

在 DynamoDB 中创建和使用表

此 Node.js 代码示例演示：

- 如何创建和管理用于存储及从 DynamoDB 检索数据的表。

情景

类似于其他数据库系统，DynamoDB 将数据存储在表中。DynamoDB 表是数据的集合，这些数据按照类似于行的项目来排列。要在 DynamoDB 中存储或访问数据，您需要创建并使用表。

在本示例中，您使用一系列 Node.js 模块对 DynamoDB 表执行基本操作。该代码使用 SDK for JavaScript，通过 `DynamoDB` 客户端类的下列方法来创建和处理表：

- `CreateTableCommand`
- `ListTablesCommand`
- `DescribeTableCommand`
- `DeleteTableCommand`

完成先决条件任务

要设置和运行此示例，请先完成以下任务：

- 设置项目环境以运行这些 Node.js 示例，并安装所需的 Amazon SDK for JavaScript 和第三方模块。请按照 GitHub 上的说明进行操作。
- 安装 SDK for JavaScript DynamoDB 客户端。有关更多信息，请参阅版本 3 中的新增功能。
- 使用用户凭证创建共享配置文件。有关提供共享凭证文件的更多信息，请参阅《Amazon SDK 和工具参考指南》中的共享配置和凭证文件。

> **Important**
>
> 这些示例使用 ECMAScript6 (ES6)。这需要使用 Node.js 版本 13.x 或更高版本。要下载并安装最新版本的 Node.js，请参阅 Node.js 下载。
> 但是，如果您更喜欢使用 CommonJS 语法，请参阅 JavaScript ES6/CommonJS 语法。

> **Note**
>
> 有关这些示例中使用的数据类型的信息，请参阅 Amazon DynamoDB 中支持的数据类型和命名规则。

创建表

创建文件名为 `create-table.js` 的 Node.js 模块。请务必如前所示配置 SDK，包括下载所需的客户端和软件包。要访问 DynamoDB，请创建一个 DynamoDB 客户端服务对象。创建一个 JSON 对象，其中包含创建表所需的参数，在本示例中包括各个属性的名称和数据类型、关键架构、表的名称以及要预配置的吞吐量单位。调用 DynamoDB 服务对象的 `CreateTableCommand` 方法。

```
import { CreateTableCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new CreateTableCommand({
    TableName: "EspressoDrinks",
    // For more information about data types,
```

```
    // see https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
HowItWorks.NamingRulesDataTypes.html#HowItWorks.DataTypes and
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
    AttributeDefinitions: [
      {
        AttributeName: "DrinkName",
        AttributeType: "S",
      },
    ],
    KeySchema: [
      {
        AttributeName: "DrinkName",
        KeyType: "HASH",
      },
    ],
    ProvisionedThroughput: {
      ReadCapacityUnits: 1,
      WriteCapacityUnits: 1,
    },
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

要运行示例，请在命令提示符中键入以下内容。

```
node create-table.js
```

此示例代码可在 GitHub 上的此处找到。

列出表

创建文件名为 `list-tables.js` 的 Node.js 模块。请务必如前所示配置 SDK，包括下载所需的客户端和软件包。要访问 DynamoDB，请创建一个 DynamoDB 客户端服务对象。创建一个 JSON 对象，其中包含列出表所需的参数，在此示例中将列出的表数量限制为 10。调用 DynamoDB 服务对象的 `ListTablesCommand` 方法。

```
import { ListTablesCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";
```

```
const client = new DynamoDBClient({});

export const main = async () => {
  const command = new ListTablesCommand({});

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

要运行示例，请在命令提示符中键入以下内容。

```
node list-tables.js
```

此示例代码可在 GitHub 上的此处找到。

描述表

创建文件名为 `describe-table.js` 的 Node.js 模块。请务必如前所示配置 SDK，包括下载所需的客户端和软件包。要访问 DynamoDB，请创建一个 DynamoDB 客户端服务对象。创建一个 JSON 对象，其中包含描述 DynamoDB 服务对象的 `DescribeTableCommand` 方法所需的参数。

```
import { DescribeTableCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new DescribeTableCommand({
    TableName: "Pastries",
  });

  const response = await client.send(command);
  console.log(`TABLE NAME: ${response.Table.TableName}`);
  console.log(`TABLE ITEM COUNT: ${response.Table.ItemCount}`);
  return response;
};
```

要运行示例，请在命令提示符中键入以下内容。

```
node describe-table.js
```

此示例代码可在 GitHub 上的此处找到。

删除表

创建文件名为 delete-table.js 的 Node.js 模块。请务必如前所示配置 SDK，包括下载所需的客户端和软件包。要访问 DynamoDB，请创建一个 DynamoDB 客户端服务对象。创建一个 JSON 对象，其中包含删除表所需的参数，在此示例中包括提供作为命令行参数的表的名称。调用 DynamoDB 服务对象的 DeleteTableCommand 方法。

```
import { DeleteTableCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new DeleteTableCommand({
    TableName: "DecafCoffees",
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

要运行示例，请在命令提示符中键入以下内容。

```
node delete-table.js
```

此示例代码可在 GitHub 上的此处找到。

在 DynamoDB 中读取和写入单个项目

此 Node.js 代码示例演示：

- 如何在 DynamoDB 表中添加项目。
- 如何在 DynamoDB 表中检索项目。
- 如何从 DynamoDB 表中删除项目。

情景

在本示例中，您使用一系列 Node.js 模块，通过 DynamoDB 客户端类的下列方法，在 DynamoDB 表中读取和写入一个项目：

- PutitemCommand
- UpdateItemCommand
- GetItemCommand
- DeleteItemCommand

完成先决条件任务

要设置和运行此示例，请先完成以下任务：

- 设置项目环境以运行这些 Node.js 示例，并安装所需的 Amazon SDK for JavaScript 和第三方模块。请按照 GitHub 上的说明进行操作。
- 使用用户凭证创建共享配置文件。有关提供共享凭证文件的更多信息，请参阅《Amazon SDK 和工具参考指南》中的共享配置和凭证文件。
- 创建一个可访问其项目的 DynamoDB 表。有关创建 DynamoDB 表的更多信息，请参阅在 DynamoDB 中创建和使用表。

> **Important**
>
> 这些示例使用 ECMAScript6 (ES6)。这需要使用 Node.js 版本 13.x 或更高版本。要下载并安装最新版本的 Node.js，请参阅 Node.js 下载。
> 但是，如果您更喜欢使用 CommonJS 语法，请参阅 JavaScript ES6/CommonJS 语法。

> **Note**
>
> 有关这些示例中使用的数据类型的信息，请参阅 Amazon DynamoDB 中支持的数据类型和命名规则。

写入项目

创建文件名为 `put-item.js` 的 Node.js 模块。请务必如前所示配置 SDK，包括下载所需的客户端和软件包。要访问 DynamoDB，请创建一个 `DynamoDB` 客户端服务对象。创建一个 JSON 对象，其中包含添加项目所需的参数，在本示例中包括表的名称，定义要设置的属性的映射，以及各个属性的值。调用 DynamoDB 客户端服务对象的 `PutItemCommand` 方法。

```
import { PutItemCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new PutItemCommand({
    TableName: "Cookies",
    // For more information about data types,
    // see https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
HowItWorks.NamingRulesDataTypes.html#HowItWorks.DataTypes and
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
    Item: {
      Flavor: { S: "Chocolate Chip" },
      Variants: { SS: ["White Chocolate Chip", "Chocolate Chunk"] },
    },
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

要运行示例，请在命令提示符中键入以下内容。

```
node put-item.js
```

此示例代码可在 GitHub 上的此处找到。

更新项目

创建文件名为 `update-item.js` 的 Node.js 模块。请务必如前所示配置 SDK，包括下载所需的客户端和软件包。要访问 DynamoDB，请创建一个 `DynamoDB` 客户端服务对象。创建一个 JSON 对象，其中包含添加项目所需的参数，在本示例中包括表的名称、要更新的密钥、与新属性名称对应的日期表达式以及各个新属性的值。调用 DynamoDB 客户端服务对象的 `UpdateItemCommand` 方法。

```
import { UpdateItemCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new UpdateItemCommand({
    TableName: "IceCreams",
    // For more information about data types,
    // see https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
HowItWorks.NamingRulesDataTypes.html#HowItWorks.DataTypes and
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
    Key: {
      Flavor: { S: "Vanilla" },
    },
    UpdateExpression: "set HasChunks = :chunks",
    ExpressionAttributeValues: {
      ":chunks": { BOOL: "false" },
    },
    ReturnValues: "ALL_NEW",
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

要运行示例，请在命令提示符中键入以下内容。

```
node update-item.js
```

此示例代码可在 GitHub 上的此处找到。

获取项目

创建文件名为 `get-item.js` 的 Node.js 模块。请务必如前所示配置 SDK，包括下载所需的客户端和软件包。要访问 DynamoDB，请创建一个 DynamoDB 客户端服务对象。要标识所需获取的项目，您必须提供该项目在表中主键的值。默认情况下，`GetItemCommand` 方法返回为项目定义的所有属性值。要仅获取所有可能属性值的子集，请指定投影表达式。

创建一个 JSON 对象，其中包含获取某个项目所需的参数，在本示例中包括表的名称，所获取项目的键的值，以及确定要检索的项目属性的投影表达式。调用 DynamoDB 客户端服务对象的 `GetItemCommand` 方法。

以下代码示例从表中检索一个项目，其主键仅由分区键组成，而不是由分区和排序键组成。如果表的主键由分区键和排序键组成，则还必须指定排序键的名称和属性。

```
import { GetItemCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new GetItemCommand({
    TableName: "CafeTreats",
    // For more information about data types,
    // see https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
HowItWorks.NamingRulesDataTypes.html#HowItWorks.DataTypes and
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
    Key: {
      TreatId: { N: "101" },
    },
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

要运行示例，请在命令提示符中键入以下内容。

```
node get-item.js
```

此示例代码可在 GitHub 上的此处找到。

删除项目

创建文件名为 `delete-item.js` 的 Node.js 模块。请务必如前所示配置 SDK，包括下载所需的客户端和软件包。要访问 DynamoDB，请创建一个 DynamoDB 客户端服务对象。创建一个 JSON 对象，其中包含删除项目所需的参数，在本示例中包括表的名称，以及所删除的项目的键名和值。调用 DynamoDB 客户端服务对象的 `DeleteItemCommand` 方法。

```
import { DeleteItemCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new DeleteItemCommand({
    TableName: "Drinks",
    // For more information about data types,
    // see https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
HowItWorks.NamingRulesDataTypes.html#HowItWorks.DataTypes and
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
    Key: {
      Name: { S: "Pumpkin Spice Latte" },
    },
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

要运行示例，请在命令提示符中键入以下内容。

```
node delete-item.js
```

此示例代码可在 GitHub 上的此处找到。

在 DynamoDB 中批量读取和写入项目

此 Node.js 代码示例演示：

- 如何在 DynamoDB 表中批量读取和写入项目。

情景

在本示例中，您使用一系列 Node.js 模块在 DynamoDB 表中批量放置项目以及批量读取项目。该代码使用 SDK for JavaScript，通过 DynamoDB 客户端类的下列方法来执行批量读取和写入操作：

- BatchGetItemCommand
- BatchWriteItemCommand

完成先决条件任务

要设置和运行此示例，请先完成以下任务：

- 设置项目环境以运行这些 Node TypeScript 示例，并安装所需的 Amazon SDK for JavaScript 和第三方模块。请按照 GitHub 上的说明进行操作。
- 使用用户凭证创建共享配置文件。有关提供共享凭证文件的更多信息，请参阅《Amazon SDK 和工具参考指南》中的共享配置和凭证文件。
- 创建一个可访问其项目的 DynamoDB 表。有关创建 DynamoDB 表的更多信息，请参阅在 DynamoDB 中创建和使用表。

> **Important**
> 这些示例使用 ECMAScript6 (ES6)。这需要使用 Node.js 版本 13.x 或更高版本。要下载并安装最新版本的 Node.js，请参阅 Node.js 下载。
> 但是，如果您更喜欢使用 CommonJS 语法，请参阅 JavaScript ES6/CommonJS 语法。

> **Note**
> 有关这些示例中使用的数据类型的信息，请参阅 Amazon DynamoDB 中支持的数据类型和命名规则。

批量读取项目

创建文件名为 `batch-get-item.js` 的 Node.js 模块。请务必如前所示配置 SDK，包括下载所需的客户端和软件包。要访问 DynamoDB，请创建一个 `DynamoDB` 客户端服务对象。创建一个 JSON 对象，其中包含批量获取项目所需的参数，在此示例中包括要读取的一个或多个表的名称，

在各个表中要读取的键的值，以及指定要返回的属性的投影表达式。调用 DynamoDB 服务对象的 BatchGetItemCommand 方法。

```
import { BatchGetItemCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new BatchGetItemCommand({
    RequestItems: {
      // Each key in this object is the name of a table. This example refers
      // to a PageAnalytics table.
      PageAnalytics: {
        // Each entry in Keys is an object that specifies a primary key.
        Keys: [
          {
            // "PageName" is the partition key (simple primary key).
            // "S" specifies a string as the data type for the value "Home".
            // For more information about data types,
            // see https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
HowItWorks.NamingRulesDataTypes.html#HowItWorks.DataTypes and
            // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
            PageName: { S: "Home" },
          },
          {
            PageName: { S: "About" },
          },
        ],
        // Only return the "PageName" and "PageViews" attributes.
        ProjectionExpression: "PageName, PageViews",
      },
    },
  });

  const response = await client.send(command);
  console.log(response.Responses["PageAnalytics"]);
  return response;
};
```

要运行示例，请在命令提示符中键入以下内容。

```
node batch-get-item.js
```

此示例代码可在 GitHub 上的此处找到。

批量写入项目

创建文件名为 batch-write-item.js 的 Node.js 模块。请务必如前所示配置 SDK，包括下载所需的客户端和软件包。要访问 DynamoDB，请创建一个 DynamoDB 客户端服务对象。创建一个 JSON 对象，其中包含批量获取项目所需的参数，在本示例中包括要写入项目的表，要写入的各个项目的键，以及属性及值。调用 DynamoDB 服务对象的 BatchWriteItemCommand 方法。

```
import {
  BatchWriteItemCommand,
  DynamoDBClient,
} from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new BatchWriteItemCommand({
    RequestItems: {
      // Each key in this object is the name of a table. This example refers
      // to a Coffees table.
      Coffees: [
        // Each entry in Coffees is an object that defines either a PutRequest or
 DeleteRequest.
        {
          // Each PutRequest object defines one item to be inserted into the table.
          PutRequest: {
            // The keys of Item are attribute names. Each attribute value is an object
 with a data type and value.
            // For more information about data types,
            // see https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
HowItWorks.NamingRulesDataTypes.html#HowItWorks.DataTypes
            Item: {
              Name: { S: "Donkey Kick" },
              Process: { S: "Wet-Hulled" },
              Flavors: { SS: ["Earth", "Syrup", "Spice"] },
            },
          },
        },
        {
```

```
          PutRequest: {
            Item: {
              Name: { S: "Flora Ethiopia" },
              Process: { S: "Washed" },
              Flavors: { SS: ["Stone Fruit", "Toasted Almond", "Delicate"] },
            },
          },
        },
      ],
    },
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

要运行示例，请在命令提示符中键入以下内容。

```
node batch-write-item.js
```

此示例代码可在 GitHub 上的此处找到。

查询并扫描 DynamoDB 表

此 Node.js 代码示例演示：

- 如何查询和扫描 DynamoDB 表的项目。

情景

仅使用主键属性值查找表或二级索引中的项目。您必须提供分区键名称和要搜索的值。您还可提供排序键名称和值，并使用比较运算符来优化搜索结果。扫描操作通过检查指定表中的每个项目来查找项目。

在本示例中，您使用一系列 Node.js 模块来标识要从 DynamoDB 表中检索的一个或多个项目。该代码使用 SDK for JavaScript，通过 DynamoDB 客户端类的下列方法来查询和扫描表：

- QueryCommand
- ScanCommand

完成先决条件任务

要设置和运行此示例，请先完成以下任务：

- 设置项目环境以运行这些 Node.js 示例，并安装所需的 Amazon SDK for JavaScript 和第三方模块。请按照 GitHub 上的说明进行操作。
- 使用用户凭证创建共享配置文件。有关提供共享凭证文件的更多信息，请参阅《Amazon SDK 和工具参考指南》中的共享配置和凭证文件。
- 创建一个可访问其项目的 DynamoDB 表。有关创建 DynamoDB 表的更多信息，请参阅在 DynamoDB 中创建和使用表。

> **Important**
>
> 这些示例使用 ECMAScript6 (ES6)。这需要使用 Node.js 版本 13.x 或更高版本。要下载并安装最新版本的 Node.js，请参阅 Node.js 下载。
> 但是，如果您更喜欢使用 CommonJS 语法，请参阅 JavaScript ES6/CommonJS 语法。
>
> > **Note**
> >
> > 有关这些示例中使用的数据类型的信息，请参阅 Amazon DynamoDB 中支持的数据类型和命名规则。

查询表

创建文件名为 `query.js` 的 Node.js 模块。请务必如前所示配置 SDK，包括下载所需的客户端和软件包。要访问 DynamoDB，请创建一个 `DynamoDB` 客户端服务对象。创建一个 JSON 对象，其中包含查询表所需的参数，在本示例中包括表名，查询所需的 `ExpressionAttributeValues`，使用这些值定义查询要返回的项目的 `KeyConditionExpression`，以及各个项目要返回的属性值的名称。调用 DynamoDB 服务对象的 `QueryCommand` 方法。

```
import { DynamoDBClient, QueryCommand } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});
```

```
export const main = async () => {
  const command = new QueryCommand({
    KeyConditionExpression: "Flavor = :flavor",
    // For more information about data types,
    // see https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
HowItWorks.NamingRulesDataTypes.html#HowItWorks.DataTypes and
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
    ExpressionAttributeValues: {
      ":flavor": { S: "Key Lime" },
      ":searchKey": { S: "no coloring" },
    },
    FilterExpression: "contains (Description, :searchKey)",
    ProjectionExpression: "Flavor, CrustType, Description",
    TableName: "Pies",
  });

  const response = await client.send(command);
  response.Items.forEach(function (pie) {
    console.log(`${pie.Flavor.S} - ${pie.Description.S}\n`);
  });
  return response;
};
```

要运行示例，请在命令提示符中键入以下内容。

```
node query.js
```

此示例代码可在 GitHub 上的此处找到。

扫描表

创建文件名为 `scan.js` 的 Node.js 模块。请务必如前所示配置 SDK，包括下载所需的客户端和软件包。要访问 DynamoDB，请创建一个 DynamoDB 客户端服务对象。创建一个 JSON 对象，其中包含扫描表中项目所需的参数，在本示例中包括表的名称，各个匹配项目要返回的属性值的列表，以及用于筛选结果集来查找包含指定短语的项目的表达式。调用 DynamoDB 服务对象的 `ScanCommand` 方法。

```
import { DynamoDBClient, ScanCommand } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
```

```
  const command = new ScanCommand({
    FilterExpression: "CrustType = :crustType",
    // For more information about data types,
    // see https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
HowItWorks.NamingRulesDataTypes.html#HowItWorks.DataTypes and
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
    ExpressionAttributeValues: {
      ":crustType": { S: "Graham Cracker" },
    },
    ProjectionExpression: "Flavor, CrustType, Description",
    TableName: "Pies",
  });

  const response = await client.send(command);
  response.Items.forEach(function (pie) {
    console.log(`${pie.Flavor.S} - ${pie.Description.S}\n`);
  });
  return response;
};
```

要运行示例，请在命令提示符中键入以下内容。

```
node scan.js
```

此示例代码可在 GitHub 上的此处找到。

使用 DynamoDB 文档客户端

此 Node.js 代码示例演示：

- 如何使用 DynamoDB 实用工具访问 DynamoDB 表。

情景

DynamoDB 文档客户端通过将属性值的概念抽象化，简化了项目的处理。此抽象化标注提供作为输入参数的原生 JavaScript 类型，以及将标注的响应数据转换为原生 JavaScript 类型。

有关 DynamoDB 文档客户端的更多信息，请参阅 GitHub 上的 @aws-sdk/lib-dynamodb 自述文件。有关使用 Amazon DynamoDB 编程的更多信息，请参阅《Amazon DynamoDB 开发人员指南》中的使用 DynamoDB 进行编程。

在本示例中，您使用一系列 Node.js 模块，通过 DynamoDB 实用工具对 DynamoDB 表执行基本操作。该代码使用 SDK for JavaScript，通过 DynamoDB 文档客户端类的下列方法来查询和扫描表：

- GetCommand
- PutCommand
- UpdateCommand
- QueryCommand
- DeleteCommand

有关配置 DynamoDB 文档客户端的更多信息，请参阅 @aws-sdk/lib-dynamodb。

先决条件任务

要设置和运行此示例，请先完成以下任务：

- 设置项目环境以运行这些 Node.js 示例，并安装所需的 Amazon SDK for JavaScript 和第三方模块。请按照 GitHub 上的说明进行操作。
- 使用用户凭证创建共享配置文件。有关提供共享凭证文件的更多信息，请参阅《Amazon SDK 和工具参考指南》中的共享配置和凭证文件。
- 创建一个可访问其项目的 DynamoDB 表。有关使用 SDK for JavaScript 创建 DynamoDB 表的更多信息，请参阅在 DynamoDB 中创建和使用表。您还可以使用 DynamoDB 控制台创建表。

> **Important**
>
> 这些示例使用 ECMAScript6 (ES6)。这需要使用 Node.js 版本 13.x 或更高版本。要下载并安装最新版本的 Node.js，请参阅 Node.js 下载。
> 但是，如果您更喜欢使用 CommonJS 语法，请参阅 JavaScript ES6/CommonJS 语法。

Note

有关这些示例中使用的数据类型的信息，请参阅 Amazon DynamoDB 中支持的数据类型和命名规则。

从表中获取项目

创建文件名为 get.js 的 Node.js 模块。确保如前所示配置 SDK，包括安装所需的客户端和软件包。其中包括 @aws-sdk/lib-dynamodb，一个向 @aws-sdk/client-dynamodb 提供文档客户端功能的库包。接下来，在创建文档客户端期间，按如下所示设置编组和解组配置（作为可选的第二个参数）。接下来，创建客户端。现在创建一个 JSON 对象，其中包含从表获取某个项目所需的参数，在本示例中包括表的名称，表中哈希键的名称，所要获取项目的哈希键的值。调用 DynamoDB 文档客户端的 GetCommand 方法。

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, GetCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new GetCommand({
    TableName: "AngryAnimals",
    Key: {
      CommonName: "Shoebill",
    },
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

要运行示例，请在命令提示符中键入以下内容。

```
node get.js
```

此示例代码可在 GitHub 上的此处找到。

将项目放入表中

创建文件名为 put.js 的 Node.js 模块。确保如前所示配置 SDK，包括安装所需的客户端和软件包。其中包括 @aws-sdk/lib-dynamodb，一个向 @aws-sdk/client-dynamodb 提供文档客户端功能的库包。接下来，在创建文档客户端期间，按如下所示设置编组和解组配置（作为可选的第二个参数）。接下来，创建客户端。创建一个 JSON 对象，其中包含将项目写入表中所需的参数，在本示例中包括表的名称，要添加或更新的项目的描述（包括哈希键和值），以及要在项目上设置的属性的名称和值。调用 DynamoDB 文档客户端的 PutCommand 方法。

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { PutCommand, DynamoDBDocumentClient } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new PutCommand({
    TableName: "HappyAnimals",
    Item: {
      CommonName: "Shiba Inu",
    },
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

要运行示例，请在命令提示符中键入以下内容。

```
node put.js
```

此示例代码可在 GitHub 上的此处找到。

更新表中的项目

创建文件名为 update.js 的 Node.js 模块。确保如前所示配置 SDK，包括安装所需的客户端和软件包。其中包括 @aws-sdk/lib-dynamodb，一个向 @aws-sdk/client-dynamodb 提供文档客户端功能的库包。接下来，在创建文档客户端期间，按如下所示设置编组和解组配置（作为可选的第二个参数）。接下来，创建客户端。创建一个 JSON 对象，其中包含将项目写入表中所需的参数，在本示例中包括表的名称，要更新的项目的键，定义要更新的项目的属性的一组 UpdateExpressions，以

及您在 ExpressionAttributeValues 参数中将值分配到的令牌。调用 DynamoDB 文档客户端的 UpdateCommand 方法。

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, UpdateCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new UpdateCommand({
    TableName: "Dogs",
    Key: {
      Breed: "Labrador",
    },
    UpdateExpression: "set Color = :color",
    ExpressionAttributeValues: {
      ":color": "black",
    },
    ReturnValues: "ALL_NEW",
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

要运行示例，请在命令提示符中键入以下内容。

```
node update.js
```

此示例代码可在 GitHub 上的此处找到。

查询表

创建文件名为 query.js 的 Node.js 模块。确保如前所示配置 SDK，包括安装所需的客户端和软件包。其中包括 @aws-sdk/lib-dynamodb，一个向 @aws-sdk/client-dynamodb 提供文档客户端功能的库包。创建一个 JSON 对象，其中包含查询表所需的参数，在本示例中包括表名，查询所需的 ExpressionAttributeValues，以及使用这些值定义查询要返回的项目的 KeyConditionExpression。调用 DynamoDB 文档客户端的 QueryCommand 方法。

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
```

```
import { QueryCommand, DynamoDBDocumentClient } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new QueryCommand({
    TableName: "CoffeeCrop",
    KeyConditionExpression:
      "OriginCountry = :originCountry AND RoastDate > :roastDate",
    ExpressionAttributeValues: {
      ":originCountry": "Ethiopia",
      ":roastDate": "2023-05-01",
    },
    ConsistentRead: true,
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

要运行示例，请在命令提示符中键入以下内容。

```
node query.js
```

此示例代码可在 GitHub 上的此处找到。

从表中删除项目

创建文件名为 `delete.js` 的 Node.js 模块。确保如前所示配置 SDK，包括安装所需的客户端和软件包。其中包括 `@aws-sdk/lib-dynamodb`，一个向 `@aws-sdk/client-dynamodb` 提供文档客户端功能的库包。接下来，在创建文档客户端期间，按如下所示设置编组和解组配置（作为可选的第二个参数）。接下来，创建客户端。要访问 DynamoDB，请创建一个 DynamoDB 对象。创建一个 JSON 对象，其中包含从表中删除某个项目所需的参数，在本示例中包括表的名称，以及所要删除项目的哈希键的名称和值。调用 DynamoDB 文档客户端的 `DeleteCommand` 方法。

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, DeleteCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);
```

```
export const main = async () => {
  const command = new DeleteCommand({
    TableName: "Sodas",
    Key: {
      Flavor: "Cola",
    },
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

要运行示例，请在命令提示符中键入以下内容。

```
node delete.js
```

此示例代码可在 GitHub 上的此处找到。

AWS Elemental MediaConvert 示例

AWS Elemental MediaConvert 是基于文件的视频转码服务与广播级的功能。您可以使用它来创建资产广播并为视频点播 (VOD) 交付整个 Internet。有关更多信息，请参阅 AWS Elemental MediaConvert 用户指南。

适用于 MediaConvert 的 JavaScript API 通过 `MediaConvert` 客户端类公开。有关更多信息，请参阅《API 参考》中的类：MediaConvert。

主题

- 获取 MediaConvert 的区域特定端点
- 在 MediaConvert 中创建和管理转码作业
- 在 MediaConvert 中使用作业模板

获取 MediaConvert 的区域特定端点

此 Node.js 代码示例演示：

- 如何从 MediaConvert 检索区域特定的端点。

情景

在此示例中，您使用 Node.js 模块调用 MediaConvert 并检索您的区域特定的端点。您可以从该服务默认端点检索您的端点 URL，因此尚不需要您的区域特定的端点。代码使用 SDK for JavaScript，通过 MediaConvert 客户端类的以下方法来检索此端点：

- `DescribeEndpointsCommand`

完成先决条件任务

要设置和运行此示例，请先完成以下任务：

- 设置项目环境以运行这些 Node TypeScript 示例，并安装所需的 Amazon SDK for JavaScript 和第三方模块。请按照 GitHub 上的说明进行操作。
- 使用用户凭证创建共享配置文件。有关提供共享凭证文件的更多信息，请参阅《Amazon SDK 和工具参考指南》中的共享配置和凭证文件。
- 创建一个 IAM 角色，该角色使 MediaConvert 能够访问输入文件以及存储输出文件的 Amazon S3 存储桶。有关更多信息，请参阅《AWS Elemental MediaConvert 用户指南》中的设置 IAM 权限。

> **Important**
>
> 此示例使用 ECMAScript6 (ES6)。这需要使用 Node.js 版本 13.x 或更高版本。要下载并安装最新版本的 Node.js，请参阅 Node.js 下载。
> 但是，如果您更喜欢使用 CommonJS 语法，请参阅 JavaScript ES6/CommonJS 语法。

获取端点 URL

创建一个 `libs` 目录，然后使用文件名 `emcClientGet.js` 创建一个 Node.js 模块。将下面的代码复制并粘贴到其中，这将创建 MediaConvert 客户端对象。将 *REGION* 替换为您的 Amazon 区域。

```
import { MediaConvertClient } from "@aws-sdk/client-mediaconvert";
// Set the AWS Region.
const REGION = "REGION";
```

```
//Set the MediaConvert Service Object
const emcClientGet = new MediaConvertClient({ region: REGION });
export { emcClientGet };
```

此示例代码可在 GitHub 上的此处找到。

创建文件名为 emc_getendpoint.js 的 Node.js 模块。确保如前所示配置 SDK，包括安装所需的客户端和软件包。

创建对象以传递 MediaConvert 客户端类的 DescribeEndpointsCommand 方法的空请求参数。然后调用 DescribeEndpointsCommand 方法。

```
// Import required AWS-SDK clients and commands for Node.js
import { DescribeEndpointsCommand } from "@aws-sdk/client-mediaconvert";
import { emcClientGet } from "./libs/emcClientGet.js";

//set the parameters.
const params = { MaxResults: 0 };

const run = async () => {
  try {
    // Create a new service object and set MediaConvert to customer endpoint
    const data = await emcClientGet.send(new DescribeEndpointsCommand(params));
    console.log("Your MediaConvert endpoint is ", data.Endpoints);
    return data;
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

要运行示例，请在命令提示符中键入以下内容。

```
node emc_getendpoint.js
```

此示例代码可在 GitHub 上的此处找到。

在 MediaConvert 中创建和管理转码作业

此 Node.js 代码示例演示：

- 如何指定区域特定的端点以用于 MediaConvert。
- 如何在 MediaConvert 中创建转码作业。
- 如何取消转码作业。
- 如何检索已完成转码作业的 JSON。
- 如何检索最多 20 个最新创建的作业的 JSON 数组。

情景

在此示例中，您使用 Node.js 模块调用 MediaConvert 来创建和管理转码作业。该代码使用 SDK for JavaScript，通过 MediaConvert 客户端类的以下方法来完成此操作：

- `CreateJobCommand`
- `CancelJobCommand`
- `GetJobCommand`
- `ListJobsCommand`

完成先决条件任务

要设置和运行此示例，请先完成以下任务：

- 设置项目环境以运行这些 Node TypeScript 示例，并安装所需的 Amazon SDK for JavaScript 和第三方模块。请按照 GitHub 上的说明进行操作。
- 使用用户凭证创建共享配置文件。有关提供共享凭证文件的更多信息，请参阅《Amazon SDK 和工具参考指南》中的共享配置和凭证文件。
- 创建和配置 Amazon S3 存储桶，提供作业输入文件和输出文件的存储。有关详细信息，请参阅《AWS Elemental MediaConvert 用户指南》中的创建用于文件的存储。
- 将输入视频上传到您为输入存储预置的 Amazon S3 存储桶。有关支持的输入视频编解码器和容器的列表，请参阅《AWS Elemental MediaConvert 用户指南》中的支持的输入编解码器和容器。

- 创建一个 IAM 角色，该角色使 MediaConvert 能够访问输入文件以及存储输出文件的 Amazon S3 存储桶。有关更多信息，请参阅《AWS Elemental MediaConvert 用户指南》中的设置 IAM 权限。

> **Important**
>
> 此示例使用 ECMAScript6 (ES6)。这需要使用 Node.js 版本 13.x 或更高版本。要下载并安装最新版本的 Node.js，请参阅 Node.js 下载。
> 但是，如果您更喜欢使用 CommonJS 语法，请参阅 JavaScript ES6/CommonJS 语法。

配置 SDK

如前所示配置 SDK，包括下载所需的客户端和软件包。由于 MediaConvert 对每个账户使用自定义端点，因此，您还必须配置 `MediaConvert` 客户端类以使用您的区域特定的端点。为此，您需要在 `mediaconvert(endpoint)` 上设置 `endpoint` 参数。

```
// Import required AWS-SDK clients and commands for Node.js
import { CreateJobCommand } from "@aws-sdk/client-mediaconvert";
import { emcClient } from "./libs/emcClient.js";
```

定义简单的转码作业

创建一个 `libs` 目录，然后使用文件名 `emcClient.js` 创建一个 Node.js 模块。将下面的代码复制并粘贴到其中，这将创建 MediaConvert 客户端对象。将 *`REGION`* 替换为您的 Amazon 区域。将 *`ENDPOINT`* 替换为您的 MediaConvert 账户端点，您可以在 MediaConvert 控制台的账户页面上获取该端点。

```
import { MediaConvertClient } from "@aws-sdk/client-mediaconvert";
// Set the account end point.
const ENDPOINT = {
  endpoint: "https://ENDPOINT_UNIQUE_STRING.mediaconvert.REGION.amazonaws.com",
};
// Set the MediaConvert Service Object
const emcClient = new MediaConvertClient(ENDPOINT);
export { emcClient };
```

此示例代码可在 GitHub 上的此处找到。

创建文件名为 `emc_createjob.js` 的 Node.js 模块。确保如前所示配置 SDK，包括安装所需的客户端和软件包。创建定义转码任务参数的 JSON。

这些参数有非常详细的说明。您可以使用 AWS Elemental MediaConvert 控制台生成 JSON 作业参数，方法是在控制台中选择您的作业设置，然后选择作业部分底部的显示作业 JSON。本示例说明了简单作业的 JSON。

> **Note**
>
> 将 *JOB_QUEUE_ARN* 替换为 M ediaConvert 作业队列，将 *IAM_ROLE_ARN* 替换为 IAM 角色的 Amazon 资源名称 (ARN)，将 *OUTPUT_BUCKET_NAME* 替换为目标存储桶名称（例如“s3://OUTPUT_BUCKET_NAME/”），将 *INPUT_BUCKET_AND_FILENAME* 替换为输入存储桶和文件名，例如“s3://INPUT_BUCKET/FILE_NAME”。

```
const params = {
  Queue: "JOB_QUEUE_ARN", //JOB_QUEUE_ARN
  UserMetadata: {
    Customer: "Amazon",
  },
  Role: "IAM_ROLE_ARN", //IAM_ROLE_ARN
  Settings: {
    OutputGroups: [
      {
        Name: "File Group",
        OutputGroupSettings: {
          Type: "FILE_GROUP_SETTINGS",
          FileGroupSettings: {
            Destination: "OUTPUT_BUCKET_NAME", //OUTPUT_BUCKET_NAME, e.g., "s3://
BUCKET_NAME/"
          },
        },
        Outputs: [
          {
            VideoDescription: {
              ScalingBehavior: "DEFAULT",
              TimecodeInsertion: "DISABLED",
              AntiAlias: "ENABLED",
              Sharpness: 50,
              CodecSettings: {
                Codec: "H_264",
                H264Settings: {
```

```
            InterlaceMode: "PROGRESSIVE",
            NumberReferenceFrames: 3,
            Syntax: "DEFAULT",
            Softness: 0,
            GopClosedCadence: 1,
            GopSize: 90,
            Slices: 1,
            GopBReference: "DISABLED",
            SlowPal: "DISABLED",
            SpatialAdaptiveQuantization: "ENABLED",
            TemporalAdaptiveQuantization: "ENABLED",
            FlickerAdaptiveQuantization: "DISABLED",
            EntropyEncoding: "CABAC",
            Bitrate: 5000000,
            FramerateControl: "SPECIFIED",
            RateControlMode: "CBR",
            CodecProfile: "MAIN",
            Telecine: "NONE",
            MinIInterval: 0,
            AdaptiveQuantization: "HIGH",
            CodecLevel: "AUTO",
            FieldEncoding: "PAFF",
            SceneChangeDetect: "ENABLED",
            QualityTuningLevel: "SINGLE_PASS",
            FramerateConversionAlgorithm: "DUPLICATE_DROP",
            UnregisteredSeiTimecode: "DISABLED",
            GopSizeUnits: "FRAMES",
            ParControl: "SPECIFIED",
            NumberBFramesBetweenReferenceFrames: 2,
            RepeatPps: "DISABLED",
            FramerateNumerator: 30,
            FramerateDenominator: 1,
            ParNumerator: 1,
            ParDenominator: 1,
          },
        },
        AfdSignaling: "NONE",
        DropFrameTimecode: "ENABLED",
        RespondToAfd: "NONE",
        ColorMetadata: "INSERT",
      },
      AudioDescriptions: [
        {
          AudioTypeControl: "FOLLOW_INPUT",
```

```
                CodecSettings: {
                  Codec: "AAC",
                  AacSettings: {
                    AudioDescriptionBroadcasterMix: "NORMAL",
                    RateControlMode: "CBR",
                    CodecProfile: "LC",
                    CodingMode: "CODING_MODE_2_0",
                    RawFormat: "NONE",
                    SampleRate: 48000,
                    Specification: "MPEG4",
                    Bitrate: 64000,
                  },
                },
                LanguageCodeControl: "FOLLOW_INPUT",
                AudioSourceName: "Audio Selector 1",
              },
            ],
            ContainerSettings: {
              Container: "MP4",
              Mp4Settings: {
                CslgAtom: "INCLUDE",
                FreeSpaceBox: "EXCLUDE",
                MoovPlacement: "PROGRESSIVE_DOWNLOAD",
              },
            },
            NameModifier: "_1",
          },
        ],
      },
    ],
    AdAvailOffset: 0,
    Inputs: [
      {
        AudioSelectors: {
          "Audio Selector 1": {
            Offset: 0,
            DefaultSelection: "NOT_DEFAULT",
            ProgramSelection: 1,
            SelectorType: "TRACK",
            Tracks: [1],
          },
        },
        VideoSelector: {
          ColorSpace: "FOLLOW",
```

```
        },
        FilterEnable: "AUTO",
        PsiControl: "USE_PSI",
        FilterStrength: 0,
        DeblockFilter: "DISABLED",
        DenoiseFilter: "DISABLED",
        TimecodeSource: "EMBEDDED",
        FileInput: "INPUT_BUCKET_AND_FILENAME", //INPUT_BUCKET_AND_FILENAME, e.g.,
 "s3://BUCKET_NAME/FILE_NAME"
      },
    ],
    TimecodeConfig: {
      Source: "EMBEDDED",
    },
  },
};
```

创建转码作业

在创建作业参数 JSON 后，调用异步 run 方法以调用 MediaConvert 客户端服务对象并传递参数。所创建作业的 ID 在响应 data 中返回。

```
const run = async () => {
  try {
    const data = await emcClient.send(new CreateJobCommand(params));
    console.log("Job created!", data);
    return data;
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

要运行示例，请在命令提示符中键入以下内容。

```
node emc_createjob.js
```

此完整示例代码可在 GitHub 上的此处找到。

取消转码作业

创建一个 libs 目录，然后使用文件名 emcClient.js 创建一个 Node.js 模块。将下面的代码复制并粘贴到其中，这将创建 MediaConvert 客户端对象。将 *REGION* 替换为您的 Amazon 区域。将 *ENDPOINT* 替换为您的 MediaConvert 账户端点，您可以在 MediaConvert 控制台的账户页面上获取该端点。

```
import { MediaConvertClient } from "@aws-sdk/client-mediaconvert";
// Set the account end point.
const ENDPOINT = {
  endpoint: "https://ENDPOINT_UNIQUE_STRING.mediaconvert.REGION.amazonaws.com",
};
// Set the MediaConvert Service Object
const emcClient = new MediaConvertClient(ENDPOINT);
export { emcClient };
```

此示例代码可在 GitHub 上的此处找到。

创建文件名为 emc_canceljob.js 的 Node.js 模块。请务必如前所示配置 SDK，包括下载所需的客户端和软件包。创建包含要取消的作业的 ID 的 JSON。然后，通过创建一个 promise 来调用 MediaConvert 客户端服务对象并传递参数，以此调用 CancelJobCommand 方法。处理 promise 回调中的响应。

Note

将 *JOB_ID* 替换为要取消的作业的 ID。

```
// Import required AWS-SDK clients and commands for Node.js
import { CancelJobCommand } from "@aws-sdk/client-mediaconvert";
import { emcClient } from "./libs/emcClient.js";

// Set the parameters
const params = { Id: "JOB_ID" }; //JOB_ID

const run = async () => {
  try {
    const data = await emcClient.send(new CancelJobCommand(params));
    console.log("Job  " + params.Id + " is canceled");
    return data;
  } catch (err) {
```

```
    console.log("Error", err);
  }
};
run();
```

要运行示例，请在命令提示符中键入以下内容。

```
node ec2_canceljob.js
```

此示例代码可在 GitHub 上的此处找到。

列出最近的转码作业

创建一个 `libs` 目录，然后使用文件名 `emcClient.js` 创建一个 Node.js 模块。将下面的代码复制并粘贴到其中，这将创建 MediaConvert 客户端对象。将 *REGION* 替换为您的 Amazon 区域。将 *ENDPOINT* 替换为您的 MediaConvert 账户端点，您可以在 MediaConvert 控制台的账户页面上获取该端点。

```
import { MediaConvertClient } from "@aws-sdk/client-mediaconvert";
// Set the account end point.
const ENDPOINT = {
  endpoint: "https://ENDPOINT_UNIQUE_STRING.mediaconvert.REGION.amazonaws.com",
};
// Set the MediaConvert Service Object
const emcClient = new MediaConvertClient(ENDPOINT);
export { emcClient };
```

此示例代码可在 GitHub 上的此处找到。

创建文件名为 `emc_listjobs.js` 的 Node.js 模块。确保如前所示配置 SDK，包括安装所需的客户端和软件包。

创建包括值的参数 JSON，这些值指定是按 `ASCENDING` 还是 `DESCENDING` 对列表排序、要检查的作业队列的 Amazon 资源名称 (ARN)，以及要包含的作业的状态。然后，通过创建一个 promise 来调用 `MediaConvert` 客户端服务对象并传递参数，以此调用 `ListJobsCommand` 方法。

> **Note**
>
> 将 *QUEUE_ARN* 替换为要检查的作业队列的 Amazon 资源名称 (ARN)，将 *STATUS* 替换为队列的状态。

```
// Import required AWS-SDK clients and commands for Node.js
import { ListJobsCommand } from "@aws-sdk/client-mediaconvert";
import { emcClient } from "./libs/emcClient.js";

// Set the parameters
const params = {
  MaxResults: 10,
  Order: "ASCENDING",
  Queue: "QUEUE_ARN",
  Status: "SUBMITTED", // e.g., "SUBMITTED"
};

const run = async () => {
  try {
    const data = await emcClient.send(new ListJobsCommand(params));
    console.log("Success. Jobs: ", data.Jobs);
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

要运行示例，请在命令提示符中键入以下内容。

```
node emc_listjobs.js
```

此示例代码可在 GitHub 上的此处找到。

在 MediaConvert 中使用作业模板

此 Node.js 代码示例演示：

- 如何创建 AWS Elemental MediaConvert 作业模板。
- 如何使用作业模板来创建转码作业。
- 如何列出您的所有作业模板。

- 如何删除作业模板。

情景

在 MediaConvert 中创建转码作业所需的 JSON 有详细说明，包含大量设置。您可以将已知工作正常的设置保存在作业模板中并用于创建以后的作业，从而节省大量时间。在此示例中，您使用 Node.js 模块调用 MediaConvert 来创建、使用和管理作业模板。该代码使用 SDK for JavaScript，通过 MediaConvert 客户端类的以下方法来完成此操作：

- `CreateJobTemplateCommand`
- `CreateJobCommand`
- `DeleteJobTemplateCommand`
- `ListJobTemplatesCommand`

完成先决条件任务

要设置和运行此示例，请先完成以下任务：

- 设置项目环境以运行这些 Node TypeScript 示例，并安装所需的 Amazon SDK for JavaScript 和第三方模块。请按照 GitHub 上的说明进行操作。
- 使用用户凭证创建共享配置文件。有关提供共享凭证文件的更多信息，请参阅《Amazon SDK 和工具参考指南》中的共享配置和凭证文件。
- 创建一个 IAM 角色，该角色使 MediaConvert 能够访问输入文件以及存储输出文件的 Amazon S3 存储桶。有关更多信息，请参阅《AWS Elemental MediaConvert 用户指南》中的设置 IAM 权限。

> **Important**
>
> 这些示例使用 ECMAScript6 (ES6)。这需要使用 Node.js 版本 13.x 或更高版本。要下载并安装最新版本的 Node.js，请参阅 Node.js 下载。
> 但是，如果您更喜欢使用 CommonJS 语法，请参阅 JavaScript ES6/CommonJS 语法。

创建作业模板

创建一个 `libs` 目录，然后使用文件名 `emcClient.js` 创建一个 Node.js 模块。将下面的代码复制并粘贴到其中，这将创建 MediaConvert 客户端对象。将 *`REGION`* 替换为您的 Amazon 区域。将

ENDPOINT 替换为您的 MediaConvert 账户端点，您可以在 MediaConvert 控制台的账户页面上获取该端点。

```
import { MediaConvertClient } from "@aws-sdk/client-mediaconvert";
// Set the account end point.
const ENDPOINT = {
  endpoint: "https://ENDPOINT_UNIQUE_STRING.mediaconvert.REGION.amazonaws.com",
};
// Set the MediaConvert Service Object
const emcClient = new MediaConvertClient(ENDPOINT);
export { emcClient };
```

此示例代码可在 GitHub 上的此处找到。

创建文件名为 `emc_create_jobtemplate.js` 的 Node.js 模块。确保如前所示配置 SDK，包括安装所需的客户端和软件包。

指定用于创建模板的参数 JSON。您可以使用来自以前成功作业中的大部分 JSON 参数来指定模板中的 `Settings` 值。此示例使用来自 在 MediaConvert 中创建和管理转码作业 的作业设置。

通过创建一个 promise 来调用 `MediaConvert` 服务对象并传递参数，以此调用 `CreateJobTemplateCommand` 方法。

Note

将 *JOB_QUEUE_ARN* 替换为要检查的作业队列的 Amazon 资源名称 (ARN)，将 *BUCKET_NAME* 替换为目标 Amazon S3 存储桶的名称，例如“s3://BUCKET_NAME/”。

```
// Import required AWS-SDK clients and commands for Node.js
import { CreateJobTemplateCommand } from "@aws-sdk/client-mediaconvert";
import { emcClient } from "./libs/emcClient.js";

const params = {
  Category: "YouTube Jobs",
  Description: "Final production transcode",
  Name: "DemoTemplate",
  Queue: "JOB_QUEUE_ARN", //JOB_QUEUE_ARN
  Settings: {
    OutputGroups: [
      {
        Name: "File Group",
```

```
        OutputGroupSettings: {
          Type: "FILE_GROUP_SETTINGS",
          FileGroupSettings: {
            Destination: "BUCKET_NAME", // BUCKET_NAME e.g., "s3://BUCKET_NAME/"
          },
        },
        Outputs: [
          {
            VideoDescription: {
              ScalingBehavior: "DEFAULT",
              TimecodeInsertion: "DISABLED",
              AntiAlias: "ENABLED",
              Sharpness: 50,
              CodecSettings: {
                Codec: "H_264",
                H264Settings: {
                  InterlaceMode: "PROGRESSIVE",
                  NumberReferenceFrames: 3,
                  Syntax: "DEFAULT",
                  Softness: 0,
                  GopClosedCadence: 1,
                  GopSize: 90,
                  Slices: 1,
                  GopBReference: "DISABLED",
                  SlowPal: "DISABLED",
                  SpatialAdaptiveQuantization: "ENABLED",
                  TemporalAdaptiveQuantization: "ENABLED",
                  FlickerAdaptiveQuantization: "DISABLED",
                  EntropyEncoding: "CABAC",
                  Bitrate: 5000000,
                  FramerateControl: "SPECIFIED",
                  RateControlMode: "CBR",
                  CodecProfile: "MAIN",
                  Telecine: "NONE",
                  MinIInterval: 0,
                  AdaptiveQuantization: "HIGH",
                  CodecLevel: "AUTO",
                  FieldEncoding: "PAFF",
                  SceneChangeDetect: "ENABLED",
                  QualityTuningLevel: "SINGLE_PASS",
                  FramerateConversionAlgorithm: "DUPLICATE_DROP",
                  UnregisteredSeiTimecode: "DISABLED",
                  GopSizeUnits: "FRAMES",
                  ParControl: "SPECIFIED",
```

```
                NumberBFramesBetweenReferenceFrames: 2,
                RepeatPps: "DISABLED",
                FramerateNumerator: 30,
                FramerateDenominator: 1,
                ParNumerator: 1,
                ParDenominator: 1,
              },
            },
            AfdSignaling: "NONE",
            DropFrameTimecode: "ENABLED",
            RespondToAfd: "NONE",
            ColorMetadata: "INSERT",
          },
          AudioDescriptions: [
            {
              AudioTypeControl: "FOLLOW_INPUT",
              CodecSettings: {
                Codec: "AAC",
                AacSettings: {
                  AudioDescriptionBroadcasterMix: "NORMAL",
                  RateControlMode: "CBR",
                  CodecProfile: "LC",
                  CodingMode: "CODING_MODE_2_0",
                  RawFormat: "NONE",
                  SampleRate: 48000,
                  Specification: "MPEG4",
                  Bitrate: 64000,
                },
              },
              LanguageCodeControl: "FOLLOW_INPUT",
              AudioSourceName: "Audio Selector 1",
            },
          ],
          ContainerSettings: {
            Container: "MP4",
            Mp4Settings: {
              CslgAtom: "INCLUDE",
              FreeSpaceBox: "EXCLUDE",
              MoovPlacement: "PROGRESSIVE_DOWNLOAD",
            },
          },
          NameModifier: "_1",
        },
      ],
```

```
      },
    ],
    AdAvailOffset: 0,
    Inputs: [
      {
        AudioSelectors: {
          "Audio Selector 1": {
            Offset: 0,
            DefaultSelection: "NOT_DEFAULT",
            ProgramSelection: 1,
            SelectorType: "TRACK",
            Tracks: [1],
          },
        },
        VideoSelector: {
          ColorSpace: "FOLLOW",
        },
        FilterEnable: "AUTO",
        PsiControl: "USE_PSI",
        FilterStrength: 0,
        DeblockFilter: "DISABLED",
        DenoiseFilter: "DISABLED",
        TimecodeSource: "EMBEDDED",
      },
    ],
    TimecodeConfig: {
      Source: "EMBEDDED",
    },
  },
};

const run = async () => {
  try {
    // Create a promise on a MediaConvert object
    const data = await emcClient.send(new CreateJobTemplateCommand(params));
    console.log("Success!", data);
    return data;
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

要运行示例，请在命令提示符中键入以下内容。

```
node emc_create_jobtemplate.js
```

此示例代码可在 GitHub 上的此处找到。

从作业模板创建转码作业

创建一个 `libs` 目录，然后使用文件名 `emcClient.js` 创建一个 Node.js 模块。将下面的代码复制并粘贴到其中，这将创建 MediaConvert 客户端对象。将 *REGION* 替换为您的 Amazon 区域。将 *ENDPOINT* 替换为您的 MediaConvert 账户端点，您可以在 MediaConvert 控制台的账户页面上获取该端点。

```
import { MediaConvertClient } from "@aws-sdk/client-mediaconvert";
// Set the account end point.
const ENDPOINT = {
  endpoint: "https://ENDPOINT_UNIQUE_STRING.mediaconvert.REGION.amazonaws.com",
};
// Set the MediaConvert Service Object
const emcClient = new MediaConvertClient(ENDPOINT);
export { emcClient };
```

此示例代码可在 GitHub 上的此处找到。

创建文件名为 `emc_template_createjob.js` 的 Node.js 模块。确保如前所示配置 SDK，包括安装所需的客户端和软件包。

创建作业创建参数 JSON，其中包括要使用的作业模板名称，以及所要使用的特定于您正在创建的作业的 `Settings`。然后，通过创建一个 promise 来调用 `MediaConvert` 客户端服务对象并传递参数，以此调用 `CreateJobsCommand` 方法。

> **Note**
>
> 将 *JOB_QUEUE_ARN* 替换为要检查的作业队列的 Amazon 资源名称 (ARN)，将 *KEY_PAIR_NAME* 替换为，将 *TEMPLATE_ NAME* 替换为，将 *ROLE_ARN* 替换为 Amazon 资源名称 (ARN)，将 *INPUT_BUCKET_AND_FILENAME* 替换为输入存储桶和文件名，例如“s3://BUCKET_NAME/FILE_NAME”。

```
// Import required AWS-SDK clients and commands for Node.js
```

```
import { CreateJobCommand } from "@aws-sdk/client-mediaconvert";
import { emcClient } from "./libs/emcClient.js";

const params = {
  Queue: "QUEUE_ARN", //QUEUE_ARN
  JobTemplate: "TEMPLATE_NAME", //TEMPLATE_NAME
  Role: "ROLE_ARN", //ROLE_ARN
  Settings: {
    Inputs: [
      {
        AudioSelectors: {
          "Audio Selector 1": {
            Offset: 0,
            DefaultSelection: "NOT_DEFAULT",
            ProgramSelection: 1,
            SelectorType: "TRACK",
            Tracks: [1],
          },
        },
        VideoSelector: {
          ColorSpace: "FOLLOW",
        },
        FilterEnable: "AUTO",
        PsiControl: "USE_PSI",
        FilterStrength: 0,
        DeblockFilter: "DISABLED",
        DenoiseFilter: "DISABLED",
        TimecodeSource: "EMBEDDED",
        FileInput: "INPUT_BUCKET_AND_FILENAME", //INPUT_BUCKET_AND_FILENAME, e.g.,
 "s3://BUCKET_NAME/FILE_NAME"
      },
    ],
  },
};

const run = async () => {
  try {
    const data = await emcClient.send(new CreateJobCommand(params));
    console.log("Success! ", data);
    return data;
  } catch (err) {
    console.log("Error", err);
  }
};
```

```
run();
```

要运行示例，请在命令提示符中键入以下内容。

```
node emc_template_createjob.js
```

此示例代码可在 GitHub 上的此处找到。

列出作业模板

创建一个 libs 目录，然后使用文件名 emcClient.js 创建一个 Node.js 模块。将下面的代码复制并粘贴到其中，这将创建 MediaConvert 客户端对象。将 *REGION* 替换为您的 Amazon 区域。将 *ENDPOINT* 替换为您的 MediaConvert 账户端点，您可以在 MediaConvert 控制台的账户页面上获取该端点。

```
import { MediaConvertClient } from "@aws-sdk/client-mediaconvert";
// Set the account end point.
const ENDPOINT = {
  endpoint: "https://ENDPOINT_UNIQUE_STRING.mediaconvert.REGION.amazonaws.com",
};
// Set the MediaConvert Service Object
const emcClient = new MediaConvertClient(ENDPOINT);
export { emcClient };
```

此示例代码可在 GitHub 上的此处找到。

创建文件名为 emc_listtemplates.js 的 Node.js 模块。确保如前所示配置 SDK，包括安装所需的客户端和软件包。

创建一个对象以传递 MediaConvert 客户端类的 listTemplates 方法的请求参数。包含值以确定要列出哪些模板（NAME、CREATION DATE、SYSTEM）、要列出多少个模板及其排序顺序。要调用 ListTemplatesCommand 方法，请创建一个 promise 来调用 MediaConvert 客户端服务对象并传递参数。

```

// Import required AWS-SDK clients and commands for Node.js
import { ListJobTemplatesCommand } from "@aws-sdk/client-mediaconvert";
import { emcClient } from "./libs/emcClient.js";

const params = {
  ListBy: "NAME",
```

```
  MaxResults: 10,
  Order: "ASCENDING",
};

const run = async () => {
  try {
    const data = await emcClient.send(new ListJobTemplatesCommand(params));
    console.log("Success ", data.JobTemplates);
    return data;
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

要运行示例，请在命令提示符中键入以下内容。

```
node emc_listtemplates.js
```

此示例代码可在 GitHub 上的此处找到。

删除作业模板

创建一个 `libs` 目录，然后使用文件名 `emcClient.js` 创建一个 Node.js 模块。将下面的代码复制并粘贴到其中，这将创建 MediaConvert 客户端对象。将 *REGION* 替换为您的 Amazon 区域。将 *ENDPOINT* 替换为您的 MediaConvert 账户端点，您可以在 MediaConvert 控制台的账户页面上获取该端点。

```
import { MediaConvertClient } from "@aws-sdk/client-mediaconvert";
// Set the account end point.
const ENDPOINT = {
  endpoint: "https://ENDPOINT_UNIQUE_STRING.mediaconvert.REGION.amazonaws.com",
};
// Set the MediaConvert Service Object
const emcClient = new MediaConvertClient(ENDPOINT);
export { emcClient };
```

此示例代码可在 GitHub 上的此处找到。

创建文件名为 `emc_deletetemplate.js` 的 Node.js 模块。确保如前所示配置 SDK，包括安装所需的客户端和软件包。

创建一个对象，以将您要删除的作业模板的名称作为 `MediaConvert` 客户端类的 `DeleteJobTemplateCommand` 方法的参数传递。要调用 `DeleteJobTemplateCommand` 方法，请创建一个 promise 来调用 MediaConvert 客户端服务对象并传递参数。

```
// Import required AWS-SDK clients and commands for Node.js
import { DeleteJobTemplateCommand } from "@aws-sdk/client-mediaconvert";
import { emcClient } from "./libs/emcClient.js";

// Set the parameters
const params = { Name: "test" }; //TEMPLATE_NAME

const run = async () => {
  try {
    const data = await emcClient.send(new DeleteJobTemplateCommand(params));
    console.log(
      "Success, template deleted! Request ID:",
      data.$metadata.requestId,
    );
    return data;
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

要运行示例，请在命令提示符中键入以下内容。

```
node emc_deletetemplate.js
```

此示例代码可在 GitHub 上的此处找到。

Amazon Lambda 示例

Amazon Lambda 是一项无服务器计算服务，可使您无需预置或管理服务器即可运行代码，创建工作负载感知型集群扩展逻辑，维护事件集成，或管理运行时。

适用于 Amazon Lambda 的 JavaScript API 通过 LambdaService 客户端类公开。

以下是演示如何借助 Amazon SDK for JavaScript v3 创建和使用 Lambda 函数的示例列表：

- 使用 API Gateway 调用 Lambda
- 创建计划事件以执行 Amazon Lambda 函数

Amazon Lex 示例

Amazon Lex 是一项用于使用语音和文本将对话界面内置到应用程序中的 Amazon 服务。

适用于 Amazon Lex 的 JavaScript API 通过 Lex Runtime Service 客户端类公开。

- 构建 Amazon Lex 聊天机器人

Amazon Polly 示例

此 Node.js 代码示例演示：

- 将使用 Amazon Polly 录制的音频上传到 Amazon S3

情景

在此示例中，将使用一系列 Node.js 模块，通过 Amazon S3 客户端类的以下方法将使用 Amazon Polly 录制的音频自动上传到 Amazon S3：

- `StartSpeechSynthesisTaskCommand`

先决条件任务

要设置和运行此示例，您必须先完成以下任务：

- 按照 GitHub 上的说明设置项目环境以运行 Node JavaScript 示例。
- 使用用户凭证创建共享配置文件。有关提供共享凭证文件的更多信息，请参阅《Amazon SDK 和工具参考指南》中的共享配置和凭证文件。
- 创建一个 Amazon Identity and Access Management (IAM) 未经身份验证的 Amazon Cognito 用户角色 polly:SynthesizeSpeech 权限，以及一个附有 IAM 角色的 Amazon Cognito 身份池。下面的使用 Amazon CloudFormation 创建 Amazon 资源部分将介绍如何创建这些资源。

> **Note**
>
> 此示例使用 Amazon Cognito，但是如果您不使用 Amazon Cognito，则您的 Amazon 用户必须具有以下 IAM 权限策略：
>
> ```
> {
>   "Version": "2012-10-17",
>   "Statement": [
>     {
>       "Action": [
>         "mobileanalytics:PutEvents",
>         "cognito-sync:*"
>       ],
>       "Resource": "*",
>       "Effect": "Allow"
>     },
>     {
>       "Action": "polly:SynthesizeSpeech",
>       "Resource": "*",
>       "Effect": "Allow"
>     }
>   ]
> }
> ```

使用 Amazon CloudFormation 创建 Amazon 资源

Amazon CloudFormation 让您能够以可预测、可重复的方式创建和预置 Amazon 基础设施部署。有关 Amazon CloudFormation 的更多信息，请参阅 Amazon CloudFormation 用户指南。

创建 Amazon CloudFormation 堆栈：

1. 按照 Amazon CLI 用户指南中的说明安装和配置 Amazon CLI。
2. 在项目文件夹的根目录中创建一个名为 `setup.yaml` 的文件，然后将 GitHub 上此处的内容复制到该文件中。

Note

Amazon CloudFormation 模板是使用 GitHub 上此处提供的 Amazon CDK 生成的。有关 Amazon CDK 的更多信息，请参阅 Amazon Cloud Development Kit (Amazon CDK) 开发人员指南。

3. 从命令行运行以下命令，将 *STACK_NAME* 替换为堆栈的唯一名称。

Important

在一个 Amazon 区域和一个 Amazon 账户中，堆栈名称必须唯一。您最多可指定 128 个字符，支持数字和连字符。

```
aws cloudformation create-stack --stack-name STACK_NAME --template-body file://
setup.yaml --capabilities CAPABILITY_IAM
```

有关 `create-stack` 命令参数的更多信息，请参阅 Amazon CLI 命令参考指南和 Amazon CloudFormation 用户指南。

4. 导航到 Amazon CloudFormation 管理控制台，选择堆栈，选择堆栈名称，然后选择资源选项卡以查看已创建资源的列表。

将使用 Amazon Polly 录制的音频上传到 Amazon S3

创建文件名为 `polly_synthesize_to_s3.js` 的 Node.js 模块。确保如前所示配置 SDK，包括安装所需的客户端和软件包。在代码中，输入 *REGION* 和 *BUCKET_NAME*。要访问 Amazon Polly，请创建一个 `Polly` 客户端服务对象。将 *"IDENTITY_POOL_ID"* 替换为您为此示例创建的 Amazon Cognito 身份池的示例页面中的 `IdentityPoolId`。这也被传递给每个客户端对象。

调用 Amazon Polly 客户端服务对象的 `StartSpeechSynthesisCommand` 方法以合成语音消息，将其上传到 Amazon S3 存储桶。

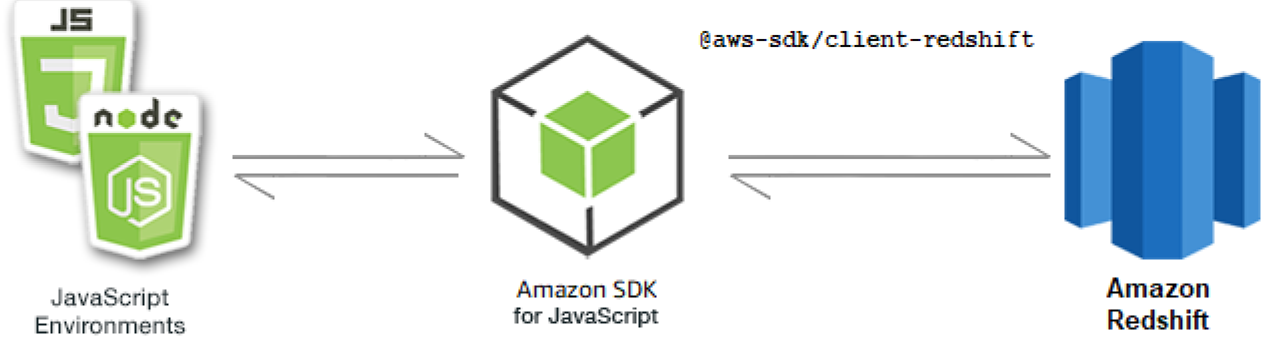
```
const { StartSpeechSynthesisTaskCommand } = require("@aws-sdk/client-polly");
const { pollyClient } = require("./libs/pollyClient.js");

// Create the parameters
var params = {
```

```
  OutputFormat: "mp3",
  OutputS3BucketName: "videoanalyzerbucket",
  Text: "Hello David, How are you?",
  TextType: "text",
  VoiceId: "Joanna",
  SampleRate: "22050",
};

const run = async () => {
  try {
    await pollyClient.send(new StartSpeechSynthesisTaskCommand(params));
    console.log("Success, audio file added to " + params.OutputS3BucketName);
  } catch (err) {
    console.log("Error putting object", err);
  }
};
run();
```

此示例代码可在 GitHub 上的此处找到。

Amazon Redshift 示例

Amazon Redshift 是一种完全托管的 PB 级云中数据仓库服务。Amazon Redshift 数据仓库是一个由称作节点的各种计算资源构成的集合，这些节点已整理到名为集群的组中。每个集群运行一个 Amazon Redshift 引擎并包含一个或多个数据库。



适用于 Amazon Redshift 的 JavaScript API 通过 Amazon Redshift 客户端类公开。

主题

- Amazon Redshift 示例

Amazon Redshift 示例

此示例使用一系列 Node.js 模块来创建、修改、描述 Amazon Redshift 集群的参数，然后使用 `Redshift` 客户端类的以下方法删除这些集群：

- `CreateClusterCommand`
- `ModifyClusterCommand`
- `DescribeClustersCommand`
- `DeleteClusterCommand`

有关 Amazon Redshift 用户的更多信息，请参阅 Amazon Redshift 入门指南。

先决条件任务

要设置和运行此示例，您必须先完成以下任务：

- 设置项目环境以运行这些 Node TypeScript 示例，并安装所需的 Amazon SDK for JavaScript 和第三方模块。请按照 GitHub 上的说明进行操作。
- 使用用户凭证创建共享配置文件。有关提供共享凭证文件的更多信息，请参阅《Amazon SDK 和工具参考指南》中的共享配置和凭证文件。

> **Important**
>
> 这些示例演示了如何使用 ECMAScript6 (ES6) 导入/导出客户端服务对象和命令。
>
> - 这需要使用 Node.js 版本 13.x 或更高版本。要下载并安装最新版本的 Node.js，请参阅 Node.js 下载。
> - 如果您更喜欢使用 CommonJS 语法，请参阅 JavaScript ES6/CommonJS 语法

创建 Amazon Redshift 集群

此示例演示如何使用 Amazon SDK for JavaScript 创建 Amazon Redshift 集群。有关更多信息，请参阅 CreateCluster。

Important

您即将创建的集群将是活跃的（且不在沙盒中运行）。您需要为该集群支付标准 Amazon Redshift 使用费，直到删除它为止。如果您在创建集群的相同位置删除了集群，则产生的总费用其实是很少的。

创建一个 `libs` 目录，然后使用文件名 `redshiftClient.js` 创建一个 Node.js 模块。将以下代码复制并粘贴到其中，这将创建 Amazon Redshift 客户端对象。将 *REGION* 替换为您的 Amazon 区域。

```
import  { RedshiftClient }  from  "@aws-sdk/client-redshift";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create Redshift service object.
const redshiftClient = new RedshiftClient({ region: REGION });
export { redshiftClient };
```

此示例代码可在 GitHub 上的此处找到。

创建文件名为 `redshift-create-cluster.js` 的 Node.js 模块。确保如前所示配置 SDK，包括安装所需的客户端和软件包。创建参数对象，指定要预置的节点类型，以及在集群中自动创建的数据库实例的主登录凭证，最后指定集群类型。

Note

将 *CLUSTER_NAME* 替换为集群的名称。对于 *NODE_TYPE*，请指定要预置的节点类型，例如“dc2.large”。*MASTER_USERNAME* 和 *MASTER_USER_PASSWORD* 是集群中数据库实例的主用户的登录凭证。对于 *CLUSTER_TYPE*，输入集群的类型。如果指定 `single-node`，则不需要 `NumberOfNodes` 参数。其余参数均为可选参数。

```
// Import required AWS SDK clients and commands for Node.js
import { CreateClusterCommand } from "@aws-sdk/client-redshift";
import { redshiftClient } from "./libs/redshiftClient.js";

const params = {
  ClusterIdentifier: "CLUSTER_NAME", // Required
  NodeType: "NODE_TYPE", //Required
  MasterUsername: "MASTER_USER_NAME", // Required - must be lowercase
```

```
  MasterUserPassword: "MASTER_USER_PASSWORD", // Required - must contain at least one
 uppercase letter, and one number
  ClusterType: "CLUSTER_TYPE", // Required
  IAMRoleARN: "IAM_ROLE_ARN", // Optional - the ARN of an IAM role with permissions
 your cluster needs to access other AWS services on your behalf, such as Amazon S3.
  ClusterSubnetGroupName: "CLUSTER_SUBNET_GROUPNAME", //Optional - the name of a
 cluster subnet group to be associated with this cluster. Defaults to 'default' if not
 specified.
  DBName: "DATABASE_NAME", // Optional - defaults to 'dev' if not specified
  Port: "PORT_NUMBER", // Optional - defaults to '5439' if not specified
};

const run = async () => {
  try {
    const data = await redshiftClient.send(new CreateClusterCommand(params));
    console.log(
      "Cluster " + data.Cluster.ClusterIdentifier + " successfully created",
    );
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

要运行示例，请在命令提示符中键入以下内容。

```
node redshift-create-cluster.js
```

此示例代码可在 GitHub 上的此处找到。

修改 Amazon Redshift 集群

此示例展示了如何使用 Amazon SDK for JavaScript 修改 Amazon Redshift 集群的主用户密码。有关您可以修改的其他设置的更多信息，请参阅 ModifyCluster。

创建一个 `libs` 目录，然后使用文件名 `redshiftClient.js` 创建一个 Node.js 模块。将以下代码复制并粘贴到其中，这将创建 Amazon Redshift 客户端对象。将 *REGION* 替换为您的 Amazon 区域。

```
import  { RedshiftClient }  from  "@aws-sdk/client-redshift";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
```

```
// Create Redshift service object.
const redshiftClient = new RedshiftClient({ region: REGION });
export { redshiftClient };
```

此示例代码可在 GitHub 上的此处找到。

创建文件名为 `redshift-modify-cluster.js` 的 Node.js 模块。确保如前所示配置 SDK，包括安装所需的客户端和软件包。指定 Amazon 区域、要修改的集群名称和新的主用户密码。

> **Note**
> 将 *CLUSTER_NAME* 替换为集群名称，将 *MASTER_USER_PASSWORD* 替换为新的主用户密码。

```
// Import required AWS SDK clients and commands for Node.js
import { ModifyClusterCommand } from "@aws-sdk/client-redshift";
import { redshiftClient } from "./libs/redshiftClient.js";

// Set the parameters
const params = {
  ClusterIdentifier: "CLUSTER_NAME",
  MasterUserPassword: "NEW_MASTER_USER_PASSWORD",
};

const run = async () => {
  try {
    const data = await redshiftClient.send(new ModifyClusterCommand(params));
    console.log("Success was modified.", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

要运行示例，请在命令提示符中键入以下内容。

```
node redshift-modify-cluster.js
```

此示例代码可在 GitHub 上的此处找到。

查看 Amazon Redshift 集群的详细信息

此示例说明了如何使用 Amazon SDK for JavaScript 查看 Amazon Redshift 集群的详细信息。有关可选内容的更多信息，请参见 DescribeClusters。

创建一个 `libs` 目录，然后使用文件名 `redshiftClient.js` 创建一个 Node.js 模块。将以下代码复制并粘贴到其中，这将创建 Amazon Redshift 客户端对象。将 *REGION* 替换为您的 Amazon 区域。

```
import  { RedshiftClient }  from  "@aws-sdk/client-redshift";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create Redshift service object.
const redshiftClient = new RedshiftClient({ region: REGION });
export { redshiftClient };
```

此示例代码可在 GitHub 上的此处找到。

创建文件名为 `redshift-describe-clusters.js` 的 Node.js 模块。确保如前所示配置 SDK，包括安装所需的客户端和软件包。指定 Amazon 区域、要修改的集群名称和新的主用户密码。

> Note
>
> 将 *CLUSTER_NAME* 替换为集群的名称。

```
// Import required AWS SDK clients and commands for Node.js
import { DescribeClustersCommand } from "@aws-sdk/client-redshift";
import { redshiftClient } from "./libs/redshiftClient.js";

const params = {
  ClusterIdentifier: "CLUSTER_NAME",
};

const run = async () => {
  try {
    const data = await redshiftClient.send(new DescribeClustersCommand(params));
    console.log("Success", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
```

```
run();
```

要运行示例，请在命令提示符中键入以下内容。

```
node redshift-describe-clusters.js
```

此示例代码可在 GitHub 上的此处找到。

删除 Amazon Redshift 集群

此示例说明了如何使用 Amazon SDK for JavaScript 查看 Amazon Redshift 集群的详细信息。有关您可以删除的其他设置的更多信息，请参阅 DeleteCluster。

创建一个 `libs` 目录，然后使用文件名 `redshiftClient.js` 创建一个 Node.js 模块。将以下代码复制并粘贴到其中，这将创建 Amazon Redshift 客户端对象。将 *REGION* 替换为您的 Amazon 区域。

```
import  { RedshiftClient }  from  "@aws-sdk/client-redshift";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create Redshift service object.
const redshiftClient = new RedshiftClient({ region: REGION });
export { redshiftClient };
```

此示例代码可在 GitHub 上的此处找到。

创建文件名为 `redshift-delete-clusters.js` 的 Node.js 模块。确保如前所示配置 SDK，包括安装所需的客户端和软件包。指定 Amazon 区域、要修改的集群名称和新的主用户密码。指定是否要在删除集群之前保存集群的最终快照，如果是，则指定快照的 ID。

> **Note**
>
> 将 *CLUSTER_NAME* 替换为集群的名称。对于 *SkipFinalClusterSnapshot*，请指定是否要在删除集群前对其创建最终快照。如果指定"false"，请在 *CLUSTER_SNAPSHOT_ID* 中指定最终集群快照的 ID。要获取此 ID，请单击集群仪表板上集群对应的快照列中的链接，然后滚动到快照窗格。请注意，词干 `rs:` 不是快照 ID 的一部分。

```
// Import required AWS SDK clients and commands for Node.js
import { DeleteClusterCommand } from "@aws-sdk/client-redshift";
import { redshiftClient } from "./libs/redshiftClient.js";
```

```
const params = {
  ClusterIdentifier: "CLUSTER_NAME",
  SkipFinalClusterSnapshot: false,
  FinalClusterSnapshotIdentifier: "CLUSTER_SNAPSHOT_ID",
};

const run = async () => {
  try {
    const data = await redshiftClient.send(new DeleteClusterCommand(params));
    console.log("Success, cluster deleted. ", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

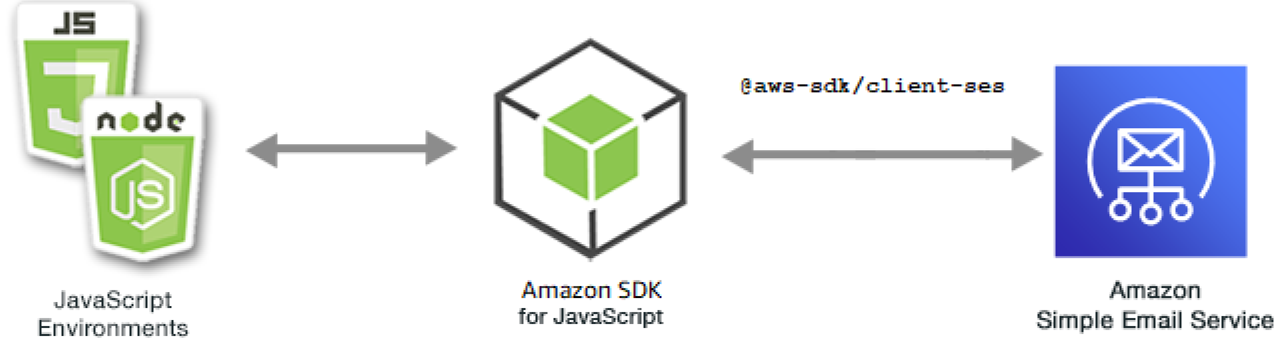
要运行示例，请在命令提示符中键入以下内容。

```
node redshift-delete-cluster.js
```

此示例代码可在 GitHub 上的此处找到。

Amazon Simple Email Service 示例

Amazon Simple Email Service（Amazon SES）是一项基于云的电子邮件发送服务，旨在帮助数字营销人员和应用程序开发人员发送营销、通知和事务电子邮件。对于使用电子邮件联系客户的所有规模的企业来说，它是一种可靠且经济实用的服务。



Amazon SES 的 JavaScript API 通过SES客户端类公开。有关使用 Amazon SES 客户端类的更多信息，请参阅《API 参考》中的类：SES。

主题

- [管理 Amazon SES 身份]
- [在 Amazon SES 中使用电子邮件模板]
- [使用 Amazon SES 发送电子邮件]

管理 Amazon SES 身份

此 Node.js 代码示例演示：

- 如何验证用于 Amazon SES 的电子邮件地址和域。
- 如何为您的 Amazon SES 身份分配 Amazon Identity and Access Management (IAM) 策略。
- 如何列出您 Amazon 账户的所有 Amazon SES 身份。
- 如何删除用于 Amazon SES 的身份。

Amazon SES 身份 是 Amazon SES 用来发送电子邮件的电子邮件地址或域。Amazon SES 要求您验证电子邮件身份，以确认您拥有该身份，并防止他人使用。

有关如何在 Amazon SES 中验证电子邮件地址和域名的详细信息，请参阅《Amazon Simple Email Service 开发人员指南》中的在 Amazon SES 中验证电子邮件地址和域。有关在 Amazon SES 中发送授权的信息，请参阅 Amazon SES 发送授权概述。

情景

在本示例中，您使用一系列 Node.js 模块验证和管理 Amazon SES 身份。Node.js 模块使用的 SDK JavaScript 来验证电子邮件地址和域名，使用SES客户端类的以下方法：

- `ListIdentitiesCommand`
- `DeleteIdentityCommand`
- `VerifyEmailIdentityCommand`
- `VerifyDomainIdentityCommand`

完成先决条件任务

要设置和运行此示例，您必须先完成以下任务：

- 设置项目环境以运行这些 Node TypeScript 示例，并安装所需的模块 Amazon SDK for JavaScript 和第三方模块。按照上的说明进行操作 GitHub。
- 使用用户凭证创建共享配置文件。有关提供共享凭证文件的更多信息，请参阅《Amazon SDK 和工具参考指南》 中的共享配置和凭证文件。

> **Important**
>
> 这些示例演示了如何使用 ECMAScript6（ES6）导入/导出客户端服务对象和命令。
>
> - 这需要使用 Node.js 版本 13.x 或更高版本。要下载并安装最新版本的 Node.js，请参阅 Node.js 下载。
> - 如果您更喜欢使用 CommonJS 语法，请参阅 JavaScript ES6/CommonJS 语法。

列出身份

在本示例中，使用 Node.js 模块列出用于 Amazon SES 的电子邮件地址和域。

创建一个 `libs` 目录，然后使用文件名 `sesClient.js` 创建一个 Node.js 模块。将以下代码复制并粘贴到其中，这将创建 Amazon SES 客户端对象。将 “*##*” 替换为您的 “ Amazon 区域”。

```
import { SESClient } from "@aws-sdk/client-ses";
// Set the AWS Region.
const REGION = "us-east-1";
// Create SES service object.
const sesClient = new SESClient({ region: REGION });
export { sesClient };
```

可以在此处找到此示例代码 GitHub。

创建文件名为 `ses_listidentities.js` 的 Node.js 模块。如前所示配置 SDK，包括安装所需的客户端和软件包。

创建对象，为 SES 客户端类的 `ListIdentitiesCommand` 方法传递 `IdentityType` 及其他参数。要调用 `ListIdentitiesCommand` 方法，请调用一个 Amazon SES 服务对象来传递参数对象。

返回的 `data` 包含 `IdentityType` 参数所指定的域身份数组。

将 *IDENTITY_TYPE* 替换为身份类型，身份类型可以是 “EmailAddress” 或 “域”。

```
import { ListIdentitiesCommand } from "@aws-sdk/client-ses";
import { sesClient } from "./libs/sesClient.js";

const createListIdentitiesCommand = () =>
  new ListIdentitiesCommand({ IdentityType: "EmailAddress", MaxItems: 10 });

const run = async () => {
  const listIdentitiesCommand = createListIdentitiesCommand();

  try {
    return await sesClient.send(listIdentitiesCommand);
  } catch (err) {
    console.log("Failed to list identities.", err);
    return err;
  }
};
```

要运行示例，请在命令提示符中键入以下内容。

```
node ses_listidentities.js
```

可以在此处找到此示例代码 GitHub。

验证电子邮件地址身份

在本示例中，使用 Node.js 模块验证用于 Amazon SES 的电子邮件发送方。

创建一个 `libs` 目录，然后使用文件名 `sesClient.js` 创建一个 Node.js 模块。将以下代码复制并粘贴到其中，这将创建 Amazon SES 客户端对象。将 “*##*” 替换为您的 “ Amazon 区域”。

```
import { SESClient } from "@aws-sdk/client-ses";
// Set the AWS Region.
const REGION = "us-east-1";
// Create SES service object.
```

```
const sesClient = new SESClient({ region: REGION });
export { sesClient };
```

可以在此处找到此示例代码 GitHub。

创建文件名为 `ses_verifyemailidentity.js` 的 Node.js 模块。如前所示配置 SDK，包括下载所需的客户端和软件包。

创建对象，为 SES 客户端类的 `VerifyEmailIdentityCommand` 方法传递 `EmailAddress` 参数。要调用 `VerifyEmailIdentityCommand` 方法，请调用一个 Amazon SES 客户端服务对象来传递参数。

> **Note**
>
> 将 *ADDRESS@DOMAIN.EXT* 替换为电子邮件地址，例如 name@example.com。

```
// Import required AWS SDK clients and commands for Node.js
import { VerifyEmailIdentityCommand } from "@aws-sdk/client-ses";
import { sesClient } from "./libs/sesClient.js";

const EMAIL_ADDRESS = "name@example.com";

const createVerifyEmailIdentityCommand = (emailAddress) => {
  return new VerifyEmailIdentityCommand({ EmailAddress: emailAddress });
};

const run = async () => {
  const verifyEmailIdentityCommand =
    createVerifyEmailIdentityCommand(EMAIL_ADDRESS);
  try {
    return await sesClient.send(verifyEmailIdentityCommand);
  } catch (err) {
    console.log("Failed to verify email identity.", err);
    return err;
  }
};
```

要运行示例，请在命令提示符中键入以下内容。域会添加到 Amazon SES 等待验证。

```
node ses_verifyemailidentity.js
```

可以在此处找到此示例代码 GitHub。

验证域身份

在本示例中，使用 Node.js 模块验证用于 Amazon SES 的电子邮件域。

创建一个 libs 目录，然后使用文件名 sesClient.js 创建一个 Node.js 模块。将以下代码复制并粘贴到其中，这将创建 Amazon SES 客户端对象。将 “*##*” 替换为您的 “ Amazon 区域”。

```
import { SESClient } from "@aws-sdk/client-ses";
// Set the AWS Region.
const REGION = "us-east-1";
// Create SES service object.
const sesClient = new SESClient({ region: REGION });
export { sesClient };
```

可以在此处找到此示例代码 GitHub。

创建文件名为 ses_verifydomainidentity.js 的 Node.js 模块。如前所示配置 SDK，包括安装所需的客户端和软件包。

创建对象，为 SES 客户端类的 VerifyDomainIdentityCommand 方法传递 Domain 参数。要调用 VerifyDomainIdentityCommand 方法，请调用一个 Amazon SES 客户端服务对象来传递参数对象。

> **Note**
>
> 此示例导入并使用所需的 S Amazon ervice V3 包客户端、V3 命令，并以异步/等待模式使用该send方法。您可以改用 V2 命令创建此示例，方法是进行一些细微的更改。有关更多信息，请参阅 使用 V3 命令。

> **Note**
>
> 将 *AMI_ID* 替换为要运行的 Amazon 机器映像 (AMI) 的 ID，以及要分配给 AMI ID 的密钥对的 *KEY_PAIR_NAME*。

```
import { VerifyDomainIdentityCommand } from "@aws-sdk/client-ses";
```

```
import {
  getUniqueName,
  postfix,
} from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "./libs/sesClient.js";

/**
 * You must have access to the domain's DNS settings to complete the
 * domain verification process.
 */
const DOMAIN_NAME = postfix(getUniqueName("Domain"), ".example.com");

const createVerifyDomainIdentityCommand = () => {
  return new VerifyDomainIdentityCommand({ Domain: DOMAIN_NAME });
};

const run = async () => {
  const VerifyDomainIdentityCommand = createVerifyDomainIdentityCommand();

  try {
    return await sesClient.send(VerifyDomainIdentityCommand);
  } catch (err) {
    console.log("Failed to verify domain.", err);
    return err;
  }
};
```

要运行示例，请在命令提示符中键入以下内容。域会添加到 Amazon SES 等待验证。

```
node ses_verifydomainidentity.js
```

可以在此处找到此示例代码 GitHub。

删除身份

在本示例中，使用 Node.js 模块删除用于 Amazon SES 的电子邮件地址或域。

创建一个 `libs` 目录，然后使用文件名 `sesClient.js` 创建一个 Node.js 模块。将以下代码复制并粘贴到其中，这将创建 Amazon SES 客户端对象。将 “*##*” 替换为您的 “ Amazon 区域”。

```
import { SESClient } from "@aws-sdk/client-ses";
// Set the AWS Region.
```

```
const REGION = "us-east-1";
// Create SES service object.
const sesClient = new SESClient({ region: REGION });
export { sesClient };
```

可以在此处找到此示例代码 GitHub。

创建文件名为 `ses_deleteidentity.js` 的 Node.js 模块。如前所示配置 SDK，包括安装所需的客户端和软件包。

创建对象，为 SES 客户端类的 `DeleteIdentityCommand` 方法传递 `Identity` 参数。要调用 `DeleteIdentityCommand` 方法，请创建一个 `request` 来调用 Amazon SES 客户端服务对象并传递参数。

> **Note**
>
> 此示例导入并使用所需的 S Amazon ervice V3 包客户端、V3 命令，并以异步/等待模式使用该send方法。您可以改用 V2 命令创建此示例，方法是进行一些细微的更改。有关更多信息，请参阅 使用 V3 命令。

> **Note**
>
> 将 *`IDENTITY_TYPE`* 替换为要删除的身份类型，将 *`IDENTITY_NAME`* 替换为要删除的身份的名称。

```
import { DeleteIdentityCommand } from "@aws-sdk/client-ses";
import { sesClient } from "./libs/sesClient.js";

const IDENTITY_EMAIL = "fake@example.com";

const createDeleteIdentityCommand = (identityName) => {
  return new DeleteIdentityCommand({
    Identity: identityName,
  });
};

const run = async () => {
  const deleteIdentityCommand = createDeleteIdentityCommand(IDENTITY_EMAIL);
```

```
  try {
    return await sesClient.send(deleteIdentityCommand);
  } catch (err) {
    console.log("Failed to delete identity.", err);
    return err;
  }
};
```

要运行示例，请在命令提示符中键入以下内容。

```
node ses_deleteidentity.js
```

可以在此处找到此示例代码 GitHub。

在 Amazon SES 中使用电子邮件模板

此 Node.js 代码示例演示：

- 如何获取所有电子邮件模板的列表。
- 如何检索和更新电子邮件模板。
- 如何创建和删除电子邮件模板。

通过 Amazon SES，您可以使用电子邮件模板发送个性化的电子邮件。有关如何在 Amazon SES 中创建和使用电子邮件模板的详细信息，请参阅《Amazon Simple Email Service 开发人员指南》中的通过 Amazon SES API 发送个性化电子邮件。

情景

在本示例中，您使用一系列 Node.js 模块来处理电子邮件模板。Node.js 模块使用的 SDK JavaScript ，使用SES客户端类的以下方法来创建和使用电子邮件模板：

- `ListTemplatesCommand`
- `CreateTemplateCommand`

- GetTemplateCommand
- DeleteTemplateCommand
- UpdateTemplateCommand

完成先决条件任务

要设置和运行此示例，您必须先完成以下任务：

- 设置项目环境以运行这些 Node TypeScript 示例，并安装所需的模块 Amazon SDK for JavaScript 和第三方模块。按照上的说明进行操作 GitHub。
- 使用用户凭证创建共享配置文件。有关提供共享凭证文件的更多信息，请参阅《Amazon SDK 和工具参考指南》 中的共享配置和凭证文件。

> **Important**
>
> 这些示例演示了如何使用 ECMAScript6（ES6）导入/导出客户端服务对象和命令。
>
> - 这需要使用 Node.js 版本 13.x 或更高版本。要下载并安装最新版本的 Node.js，请参阅 Node.js 下载。
> - 如果您更喜欢使用 CommonJS 语法，请参阅 JavaScript ES6/CommonJS 语法。

列出电子邮件模板

在本示例中，使用 Node.js 模块创建用于 Amazon SES 的电子邮件模板。

创建一个 `libs` 目录，然后使用文件名 `sesClient.js` 创建一个 Node.js 模块。将以下代码复制并粘贴到其中，这将创建 Amazon SES 客户端对象。将 “##” 替换为您的 “ Amazon 区域”。

```
import { SESClient } from "@aws-sdk/client-ses";
// Set the AWS Region.
const REGION = "us-east-1";
// Create SES service object.
const sesClient = new SESClient({ region: REGION });
export { sesClient };
```

可以在此处找到此示例代码 GitHub。

创建文件名为 `ses_listtemplates.js` 的 Node.js 模块。如前所示配置 SDK，包括安装所需的客户端和软件包。

创建对象，为 SES 客户端类的 `ListTemplatesCommand` 方法传递参数。要调用 `ListTemplatesCommand` 方法，请调用一个 Amazon SES 客户端服务对象来传递参数。

> **Note**
>
> 此示例导入并使用所需的 S Amazon ervice V3 包客户端、V3 命令，并以异步/等待模式使用该`send`方法。您可以改用 V2 命令创建此示例，方法是进行一些细微的更改。有关更多信息，请参阅 使用 V3 命令。

> **Note**
>
> 将 *ITEMS_COUNT* 替换为要返回的最大模板数。取值范围为 1-10。

```
import { ListTemplatesCommand } from "@aws-sdk/client-ses";
import { sesClient } from "./libs/sesClient.js";

const createListTemplatesCommand = (maxItems) =>
  new ListTemplatesCommand({ MaxItems: maxItems });

const run = async () => {
  const listTemplatesCommand = createListTemplatesCommand(10);

  try {
    return await sesClient.send(listTemplatesCommand);
  } catch (err) {
    console.log("Failed to list templates.", err);
    return err;
  }
};
```

要运行示例，请在命令提示符中键入以下内容。Amazon SES 会返回模板列表。

```
node ses_listtemplates.js
```

可以在此处找到此示例代码 GitHub。

获取电子邮件模板

在本示例中，使用 Node.js 模块获取用于 Amazon SES 的电子邮件模板。

创建一个 `libs` 目录，然后使用文件名 `sesClient.js` 创建一个 Node.js 模块。将以下代码复制并粘贴到其中，这将创建 Amazon SES 客户端对象。将 “*##*” 替换为您的 “ Amazon 区域”。

```
import { SESClient } from "@aws-sdk/client-ses";
// Set the AWS Region.
const REGION = "us-east-1";
// Create SES service object.
const sesClient = new SESClient({ region: REGION });
export { sesClient };
```

可以在此处找到此示例代码 GitHub。

创建文件名为 `ses_gettemplate.js` 的 Node.js 模块。如前所示配置 SDK，包括安装所需的客户端和软件包。

创建对象，为 SES 客户端类的 `GetTemplateCommand` 方法传递 `TemplateName` 参数。要调用 `GetTemplateCommand` 方法，请调用一个 Amazon SES 客户端服务对象来传递参数。

> **Note**
>
> 此示例导入并使用所需的 S Amazon ervice V3 包客户端、V3 命令，并以异步/等待模式使用该send方法。您可以改用 V2 命令创建此示例，方法是进行一些细微的更改。有关更多信息，请参阅 使用 V3 命令。

> **Note**
>
> 将 *TEMPLATE_NAME* 替换为要返回的模板的名称。

```
import { GetTemplateCommand } from "@aws-sdk/client-ses";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "./libs/sesClient.js";

const TEMPLATE_NAME = getUniqueName("TemplateName");
```

```
const createGetTemplateCommand = (templateName) =>
  new GetTemplateCommand({ TemplateName: templateName });

const run = async () => {
  const getTemplateCommand = createGetTemplateCommand(TEMPLATE_NAME);

  try {
    return await sesClient.send(getTemplateCommand);
  } catch (err) {
    console.log("Failed to get email template.", err);
    return err;
  }
};
```

要运行示例，请在命令提示符中键入以下内容。Amazon SES 会返回模板详细信息。

```
node ses_gettemplate.js
```

可以在此处找到此示例代码 GitHub。

创建电子邮件模板

在本示例中，使用 Node.js 模块创建用于 Amazon SES 的电子邮件模板。

创建一个 `libs` 目录，然后使用文件名 `sesClient.js` 创建一个 Node.js 模块。将以下代码复制并粘贴到其中，这将创建 Amazon SES 客户端对象。将 "*##*" 替换为您的 " Amazon 区域"。

```
import { SESClient } from "@aws-sdk/client-ses";
// Set the AWS Region.
const REGION = "us-east-1";
// Create SES service object.
const sesClient = new SESClient({ region: REGION });
export { sesClient };
```

可以在此处找到此示例代码 GitHub。

创建文件名为 `ses_createtemplate.js` 的 Node.js 模块。如前所示配置 SDK，包括安装所需的客户端和软件包。

创建一个对象来为 SES 客户端类的 `CreateTemplateCommand` 方法传递参数，其中包括 `TemplateName`、`HtmlPart`、`SubjectPart` 和 `TextPart`。要调用 `CreateTemplateCommand` 方法，请调用一个 Amazon SES 客户端服务对象来传递参数。

> **Note**
>
> 此示例导入并使用所需的 S Amazon ervice V3 包客户端、V3 命令，并以异步/等待模式使用该send方法。您可以改用 V2 命令创建此示例，方法是进行一些细微的更改。有关更多信息，请参阅 使用 V3 命令。

> **Note**
>
> 此示例导入并使用所需的 S Amazon ervice V3 包客户端、V3 命令，并以异步/等待模式使用该send方法。您可以改用 V2 命令创建此示例，方法是进行一些细微的更改。有关更多信息，请参阅 使用 V3 命令。

> **Note**
>
> 将 *TEMPLATE_NAME* 替换为新模板的名称，将 *HTML_CONTENT* 替换为带有 HTML 标签的电子邮件内容，将 *SUBJECT* 替换为电子邮件的主题，将 *TEXT_CONTENT* 替换为电子邮件的文本。

```
import { CreateTemplateCommand } from "@aws-sdk/client-ses";
import { sesClient } from "./libs/sesClient.js";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";

const TEMPLATE_NAME = getUniqueName("TestTemplateName");

const createCreateTemplateCommand = () => {
  return new CreateTemplateCommand({
    /**
     * The template feature in Amazon SES is based on the Handlebars template system.
     */
    Template: {
      /**
       * The name of an existing template in Amazon SES.
       */
      TemplateName: TEMPLATE_NAME,
      HtmlPart: `
        <h1>Hello, {{contact.firstName}}!</h1>
```

```
        <p>
        Did you know Amazon has a mascot named Peccy?
        </p>
      `,
      SubjectPart: "Amazon Tip",
    },
  });
};

const run = async () => {
  const createTemplateCommand = createCreateTemplateCommand();

  try {
    return await sesClient.send(createTemplateCommand);
  } catch (err) {
    console.log("Failed to create template.", err);
    return err;
  }
};
```

要运行示例，请在命令提示符中键入以下内容。该模板已添加到 Amazon SES。

```
node ses_createtemplate.js
```

可以在此处找到此示例代码 GitHub。

更新电子邮件模板

在本示例中，使用 Node.js 模块创建用于 Amazon SES 的电子邮件模板。

创建一个 `libs` 目录，然后使用文件名 `sesClient.js` 创建一个 Node.js 模块。将以下代码复制并粘贴到其中，这将创建 Amazon SES 客户端对象。将 "*##*" 替换为您的 " Amazon 区域"。

```
import { SESClient } from "@aws-sdk/client-ses";
// Set the AWS Region.
const REGION = "us-east-1";
// Create SES service object.
const sesClient = new SESClient({ region: REGION });
export { sesClient };
```

可以在此处找到此示例代码 GitHub。

创建文件名为 `ses_updatetemplate.js` 的 Node.js 模块。如前所示配置 SDK，包括安装所需的客户端和软件包。

创建一个对象来传递您在模板中要更新的 `Template` 参数值，并将必需的 `TemplateName` 参数传递到 SES 客户端类的 `UpdateTemplateCommand` 方法。要调用 `UpdateTemplateCommand` 方法，请调用一个 Amazon SES 服务对象来传递参数。

> **Note**
>
> 此示例导入并使用所需的 S Amazon ervice V3 包客户端、V3 命令，并以异步/等待模式使用该`send`方法。您可以改用 V2 命令创建此示例，方法是进行一些细微的更改。有关更多信息，请参阅 使用 V3 命令。

> **Note**
>
> 将 *TEMPLATE_NAME* 替换为模板的名称，将 *HTML_CONTENT* 替换为带有 HTML 标签的电子邮件内容，将 *SUBJECT* 替换为电子邮件的主题，将 *TEXT_CONTENT* 替换为电子邮件的文本。

```
import { UpdateTemplateCommand } from "@aws-sdk/client-ses";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "./libs/sesClient.js";

const TEMPLATE_NAME = getUniqueName("TemplateName");
const HTML_PART = "<h1>Hello, World!</h1>";

const createUpdateTemplateCommand = () => {
  return new UpdateTemplateCommand({
    Template: {
      TemplateName: TEMPLATE_NAME,
      HtmlPart: HTML_PART,
      SubjectPart: "Example",
      TextPart: "Updated template text.",
    },
  });
};

const run = async () => {
  const updateTemplateCommand = createUpdateTemplateCommand();
```

```
  try {
    return await sesClient.send(updateTemplateCommand);
  } catch (err) {
    console.log("Failed to update template.", err);
    return err;
  }
};
```

要运行示例，请在命令提示符中键入以下内容。Amazon SES 会返回模板详细信息。

```
node ses_updatetemplate.js
```

可以在此处找到此示例代码 GitHub。

删除电子邮件模板

在本示例中，使用 Node.js 模块创建用于 Amazon SES 的电子邮件模板。

创建一个 `libs` 目录，然后使用文件名 `sesClient.js` 创建一个 Node.js 模块。将以下代码复制并粘贴到其中，这将创建 Amazon SES 客户端对象。将 "*##*" 替换为您的 " Amazon 区域"。

```
import { SESClient } from "@aws-sdk/client-ses";
// Set the AWS Region.
const REGION = "us-east-1";
// Create SES service object.
const sesClient = new SESClient({ region: REGION });
export { sesClient };
```

可以在此处找到此示例代码 GitHub。

创建文件名为 `ses_deletetemplate.js` 的 Node.js 模块。如前所示配置 SDK，包括安装所需的客户端和软件包。

创建对象，将必需的 `TemplateName` 参数传递到 SES 客户端类的 `DeleteTemplateCommand` 方法。要调用 `DeleteTemplateCommand` 方法，请调用一个 Amazon SES 服务对象来传递参数。

> **Note**
>
> 此示例导入并使用所需的 S Amazon ervice V3 包客户端、V3 命令，并以异步/等待模式使用该send方法。您可以改用 V2 命令创建此示例，方法是进行一些细微的更改。有关更多信息，请参阅 使用 V3 命令。

Note

将 *TEMPLATE_NAME* 替换为要删除的模板的名称。

```
import { DeleteTemplateCommand } from "@aws-sdk/client-ses";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "./libs/sesClient.js";

const TEMPLATE_NAME = getUniqueName("TemplateName");

const createDeleteTemplateCommand = (templateName) =>
  new DeleteTemplateCommand({ TemplateName: templateName });

const run = async () => {
  const deleteTemplateCommand = createDeleteTemplateCommand(TEMPLATE_NAME);

  try {
    return await sesClient.send(deleteTemplateCommand);
  } catch (err) {
    console.log("Failed to delete template.", err);
    return err;
  }
};
```

要运行示例，请在命令提示符中键入以下内容。Amazon SES 会返回模板详细信息。

```
node ses_deletetemplate.js
```

可以在此处找到此示例代码 GitHub。

使用 Amazon SES 发送电子邮件

此 Node.js 代码示例演示：

- 发送文本或 HTML 电子邮件。

- 根据电子邮件模板发送电子邮件。
- 根据电子邮件模板批量发送电子邮件。

Amazon SES API 为您提供了两种不同的方法来发送电子邮件，具体取决于您对电子邮件内容的控制程度：格式化和原始。有关详细信息，请参阅使用 Amazon SES API 发送格式化电子邮件和使用 Amazon SES API 发送原始电子邮件。

情景

在本示例中，您使用一系列 Node.js 模块以多种方式发送电子邮件。Node.js 模块使用的 SDK JavaScript ，使用SES客户端类的以下方法来创建和使用电子邮件模板：

- `SendEmailCommand`
- `SendTemplatedEmailCommand`
- `SendBulkTemplatedEmailCommand`

完成先决条件任务

要设置和运行此示例，您必须先完成以下任务：

- 设置项目环境以运行这些 Node TypeScript 示例，并安装所需的模块 Amazon SDK for JavaScript 和第三方模块。按照上的说明进行操作 GitHub。
- 使用用户凭证创建共享配置文件。有关提供共享凭证文件的更多信息，请参阅《Amazon SDK 和工具参考指南》 中的共享配置和凭证文件。

> Important
>
> 这些示例演示了如何使用 ECMAScript6（ES6）导入/导出客户端服务对象和命令。
>
> - 这需要使用 Node.js 版本 13.x 或更高版本。要下载并安装最新版本的 Node.js，请参阅 Node.js 下载。
> - 如果您更喜欢使用 CommonJS 语法，请参阅 JavaScript ES6/CommonJS 语法。

电子邮件发送要求

Amazon SES 编写电子邮件并立即将其加入队列等待发送。要使用 `SendEmailCommand` 方法发送电子邮件，您的邮件必须满足以下要求：

- 您必须从已验证的电子邮件地址或域发送邮件。如果您尝试使用未验证的地址或域发送电子邮件，则操作会导致 `"Email address not verified"` 错误。
- 如果您的账户仍在 Amazon SES 沙盒中，则只能发送到经验证的地址或域，或者与 Amazon SES 邮箱模拟器关联的电子邮件地址。有关更多信息，请参阅《Amazon Simple Email Service 开发人员指南》中的验证电子邮件地址和域。
- 邮件（包括附件）的总大小必须小于 10 MB。
- 邮件必须包含至少一个收件人电子邮件地址。收件人地址可以是“收件人：”地址、“抄送：”地址或“密件抄送：”地址。如果某个收件人的电子邮件地址无效（即，未使用格式 `UserName@[SubDomain.]Domain.TopLevelDomain`），则将拒绝整个邮件，即使邮件包含的其他收件人有效。
- 邮件在“收件人：”、“抄送：”和“密件抄送：”字段中包含的收件人不能超过 50 个。如果您需要将电子邮件发送给更多的受众，可以将收件人列表划分为不超过 50 个人的组，然后多次调用 `sendEmail` 方法来发送邮件到各个组。

发送电子邮件

在本示例中，使用 Node.js 模块通过 Amazon SES 发送电子邮件。

创建一个 `libs` 目录，然后使用文件名 `sesClient.js` 创建一个 Node.js 模块。将以下代码复制并粘贴到其中，这将创建 Amazon SES 客户端对象。将 “*##*” 替换为您的 “ Amazon 区域”。

```
import { SESClient } from "@aws-sdk/client-ses";
// Set the AWS Region.
const REGION = "us-east-1";
// Create SES service object.
const sesClient = new SESClient({ region: REGION });
export { sesClient };
```

可以在此处找到此示例代码 GitHub。

创建文件名为 `ses_sendemail.js` 的 Node.js 模块。如前所示配置 SDK，包括安装所需的客户端和软件包。

创建一个对象以将定义要发送的电子邮件的参数值传递到 SES 客户端类的 `SendEmailCommand` 方法，这些参数值包括发件人和收件人地址、主题、纯文本和 HTML 格式的电子邮件正文。要调用 `SendEmailCommand` 方法，请调用一个 Amazon SES 服务对象来传递参数。

> **Note**
>
> 此示例导入并使用所需的 S Amazon ervice V3 包客户端、V3 命令，并以异步/等待模式使用该`send`方法。您可以改用 V2 命令创建此示例，方法是进行一些细微的更改。有关更多信息，请参阅 使用 V3 命令。

> **Note**
>
> 将 *RECEIVER_ADDRESS* 替换为要将电子邮件发送到的地址，将 *SENDER_ADDRESS* 替换为发送电子邮件的电子邮件地址。

```
import { SendEmailCommand } from "@aws-sdk/client-ses";
import { sesClient } from "./libs/sesClient.js";

const createSendEmailCommand = (toAddress, fromAddress) => {
  return new SendEmailCommand({
    Destination: {
      /* required */
      CcAddresses: [
        /* more items */
      ],
      ToAddresses: [
        toAddress,
        /* more To-email addresses */
      ],
    },
    Message: {
      /* required */
      Body: {
        /* required */
        Html: {
          Charset: "UTF-8",
          Data: "HTML_FORMAT_BODY",
        },
        Text: {
```

```
          Charset: "UTF-8",
          Data: "TEXT_FORMAT_BODY",
        },
      },
      Subject: {
        Charset: "UTF-8",
        Data: "EMAIL_SUBJECT",
      },
    },
    Source: fromAddress,
    ReplyToAddresses: [
      /* more items */
    ],
  });
};

const run = async () => {
  const sendEmailCommand = createSendEmailCommand(
    "recipient@example.com",
    "sender@example.com",
  );

  try {
    return await sesClient.send(sendEmailCommand);
  } catch (e) {
    console.error("Failed to send email.");
    return e;
  }
};
```

要运行示例，请在命令提示符中键入以下内容。电子邮件会排队，等待由 Amazon SES 发送。

```
node ses_sendemail.js
```

可以在此处找到此示例代码 GitHub。

使用模板发送电子邮件

在本示例中，使用 Node.js 模块通过 Amazon SES 发送电子邮件。创建文件名为 `ses_sendtemplatedemail.js` 的 Node.js 模块。如前所示配置 SDK，包括安装所需的客户端和软件包。

创建一个对象以将定义要发送的电子邮件的参数值传递到 SES 客户端类的 `SendTemplatedEmailCommand` 方法，这些参数值包括发件人和收件人地址、主题、纯文本和 HTML 格式的电子邮件正文。要调用 `SendTemplatedEmailCommand` 方法，请调用一个 Amazon SES 客户端服务对象来传递参数。

> **Note**
>
> 此示例导入并使用所需的 S Amazon ervice V3 包客户端、V3 命令，并以异步/等待模式使用该send方法。您可以改用 V2 命令创建此示例，方法是进行一些细微的更改。有关更多信息，请参阅 使用 V3 命令。

> **Note**
>
> *# REGION ######## Amazon ### RECEIVER_ ADDRESS ############### SENDER_ ADDRESS ################## TEMPLATE_NAME ########*

```
import { SendTemplatedEmailCommand } from "@aws-sdk/client-ses";
import {
  getUniqueName,
  postfix,
} from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "./libs/sesClient.js";

/**
 * Replace this with the name of an existing template.
 */
const TEMPLATE_NAME = getUniqueName("ReminderTemplate");

/**
 * Replace these with existing verified emails.
 */
const VERIFIED_EMAIL = postfix(getUniqueName("Bilbo"), "@example.com");

const USER = { firstName: "Bilbo", emailAddress: VERIFIED_EMAIL };

/**
 *
 * @param { { emailAddress: string, firstName: string } } user
 * @param { string } templateName - The name of an existing template in Amazon SES.
```

```
 * @returns { SendTemplatedEmailCommand }
 */
const createReminderEmailCommand = (user, templateName) => {
  return new SendTemplatedEmailCommand({
    /**
     * Here's an example of how a template would be replaced with user data:
     * Template: <h1>Hello {{contact.firstName}},</h1><p>Don't forget about the party
 gifts!</p>
     * Destination: <h1>Hello Bilbo,</h1><p>Don't forget about the party gifts!</p>
     */
    Destination: { ToAddresses: [user.emailAddress] },
    TemplateData: JSON.stringify({ contact: { firstName: user.firstName } }),
    Source: VERIFIED_EMAIL,
    Template: templateName,
  });
};

const run = async () => {
  const sendReminderEmailCommand = createReminderEmailCommand(
    USER,
    TEMPLATE_NAME,
  );
  try {
    return await sesClient.send(sendReminderEmailCommand);
  } catch (err) {
    console.log("Failed to send template email", err);
    return err;
  }
};
```

要运行示例，请在命令提示符中键入以下内容。电子邮件会排队，等待由 Amazon SES 发送。

```
node ses_sendtemplatedemail.js
```

可以在此处找到此示例代码 GitHub。

使用模板批量发送电子邮件

在本示例中，使用 Node.js 模块通过 Amazon SES 发送电子邮件。

创建一个 `libs` 目录，然后使用文件名 `sesClient.js` 创建一个 Node.js 模块。将以下代码复制并粘贴到其中，这将创建 Amazon SES 客户端对象。将 “*##*” 替换为您的 “ Amazon 区域”。

```
import { SESClient } from "@aws-sdk/client-ses";
// Set the AWS Region.
const REGION = "us-east-1";
// Create SES service object.
const sesClient = new SESClient({ region: REGION });
export { sesClient };
```

可以在此处找到此示例代码 GitHub。

创建文件名为 `ses_sendbulktemplatedemail.js` 的 Node.js 模块。如前所示配置 SDK，包括安装所需的客户端和软件包。

创建一个对象以将定义要发送的电子邮件的参数值传递到 SES 客户端类的 `SendBulkTemplatedEmailCommand` 方法，这些参数值包括发件人和收件人地址、主题、纯文本和 HTML 格式的电子邮件正文。要调用 `SendBulkTemplatedEmailCommand` 方法，请调用一个 Amazon SES 服务对象来传递参数。

> **Note**
>
> 此示例导入并使用所需的 S Amazon ervice V3 包客户端、V3 命令，并以异步/等待模式使用该send方法。您可以改用 V2 命令创建此示例，方法是进行一些细微的更改。有关更多信息，请参阅 使用 V3 命令。

> **Note**
>
> 将 *RECEIVER_ADDRESSES* 替换为要将电子邮件发送到的地址，将 *SENDER_ADDRESS* 替换为发送电子邮件的电子邮件地址。

```
import { SendBulkTemplatedEmailCommand } from "@aws-sdk/client-ses";
import {
  getUniqueName,
  postfix,
} from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "./libs/sesClient.js";

/**
 * Replace this with the name of an existing template.
 */
```

```
const TEMPLATE_NAME = getUniqueName("ReminderTemplate");

/**
 * Replace these with existing verified emails.
 */
const VERIFIED_EMAIL_1 = postfix(getUniqueName("Bilbo"), "@example.com");
const VERIFIED_EMAIL_2 = postfix(getUniqueName("Frodo"), "@example.com");

const USERS = [
  { firstName: "Bilbo", emailAddress: VERIFIED_EMAIL_1 },
  { firstName: "Frodo", emailAddress: VERIFIED_EMAIL_2 },
];

/**
 *
 * @param { { emailAddress: string, firstName: string }[] } users
 * @param { string } templateName the name of an existing template in SES
 * @returns { SendBulkTemplatedEmailCommand }
 */
const createBulkReminderEmailCommand = (users, templateName) => {
  return new SendBulkTemplatedEmailCommand({
    /**
     * Each 'Destination' uses a corresponding set of replacement data. We can map each
 user
     * to a 'Destination' and provide user specific replacement data to create
 personalized emails.
     *
     * Here's an example of how a template would be replaced with user data:
     * Template: <h1>Hello {{name}},</h1><p>Don't forget about the party gifts!</p>
     * Destination 1: <h1>Hello Bilbo,</h1><p>Don't forget about the party gifts!</p>
     * Destination 2: <h1>Hello Frodo,</h1><p>Don't forget about the party gifts!</p>
     */
    Destinations: users.map((user) => ({
      Destination: { ToAddresses: [user.emailAddress] },
      ReplacementTemplateData: JSON.stringify({ name: user.firstName }),
    })),
    DefaultTemplateData: JSON.stringify({ name: "Shireling" }),
    Source: VERIFIED_EMAIL_1,
    Template: templateName,
  });
};

const run = async () => {
  const sendBulkTemplateEmailCommand = createBulkReminderEmailCommand(
```

```
    USERS,
    TEMPLATE_NAME,
  );
  try {
    return await sesClient.send(sendBulkTemplateEmailCommand);
  } catch (err) {
    console.log("Failed to send bulk template email", err);
    return err;
  }
};
```

要运行示例，请在命令提示符中键入以下内容。电子邮件会排队，等待由 Amazon SES 发送。

```
node ses_sendbulktemplatedemail.js
```

可以在此处找到此示例代码 GitHub。

Amazon Simple Notification Service 示例

Amazon Simple Notification Service (Amazon SNS) 是一项 Web 服务，用于协调和管理向订阅端点或客户端交付或发送消息的过程。

在 Amazon SNS 中有两种类型的客户端：发布者和订阅者，也称为生产者和消费者。

发布者通过创建消息并将消息发送至主题与订阅者进行异步交流，主题是一个逻辑访问点和通信渠道。订阅者（即 Web 服务器、电子邮件地址、Amazon SQS 队列、Amazon Lambda 函数）在其订阅主题后通过受支持协议（Amazon SQS、HTTP/S、电子邮件、SMS、Amazon Lambda）中的一种来使用或接收邮件或通知。

适用于 Amazon SNS 的 JavaScript API 通过 Class: SNS 公开。

主题

- 在 Amazon SNS 中管理主题
- 在 Amazon SNS 中发布消息
- 在 Amazon SNS 中管理订阅
- 使用 Amazon SNS 发送 SMS 消息

在 Amazon SNS 中管理主题

此 Node.js 代码示例演示：

- 如何在 Amazon SNS 中创建可以将通知发布到的主题。
- 如何删除在 Amazon SNS 中创建的主题。
- 如何获取可用主题的列表。
- 如何获取和设置主题属性。

情景

在本示例中，您使用一系列 Node.js 模块来创建、列出和删除 Amazon SNS 主题，以及处理主题属性。Node.js 模块使用 SDK for JavaScript，通过 SNS 客户端类的以下方法管理主题：

- `CreateTopicCommand`
- `ListTopicsCommand`
- `DeleteTopicCommand`
- `GetTopicAttributesCommand`
- `SetTopicAttributesCommand`

先决条件任务

要设置和运行此示例，您必须先完成以下任务：

- 设置项目环境以运行这些 Node TypeScript 示例，并安装所需的 Amazon SDK for JavaScript 和第三方模块。请按照 GitHub 上的说明进行操作。
- 使用用户凭证创建共享配置文件。有关提供共享凭证文件的更多信息，请参阅《Amazon SDK 和工具参考指南》 中的共享配置和凭证文件。

> **Important**
> 这些示例演示了如何使用 ECMAScript6 (ES6) 导入/导出客户端服务对象和命令。

- 这需要使用 Node.js 版本 13.x 或更高版本。要下载并安装最新版本的 Node.js，请参阅 Node.js 下载。
- 如果您更喜欢使用 CommonJS 语法，请参阅 JavaScript ES6/CommonJS 语法。

创建主题

在本示例中，使用 Node.js 模块创建 Amazon SNS 主题。

创建一个 `libs` 目录，然后使用文件名 `snsClient.js` 创建一个 Node.js 模块。将以下代码复制并粘贴到其中，这将创建 Amazon SNS 客户端对象。将 *REGION* 替换为您的 Amazon 区域。

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

此示例代码可在 GitHub 上的此处找到。

创建文件名为 `create-topic.js` 的 Node.js 模块。如前所示配置 SDK，包括安装所需的客户端和软件包。

创建对象，将新主题的 `Name` 传递到 SNS 客户端类的 `CreateTopicCommand` 方法。要调用 `CreateTopicCommand` 方法，请创建一个用于调用 Amazon SNS 服务对象的异步函数并传递参数对象。返回的 `data` 包含主题的 ARN。

> **Note**
>
> 将 *TOPIC_NAME* 替换为主题名称。

```
import { CreateTopicCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicName - The name of the topic to create.
```

```
 */
export const createTopic = async (topicName = "TOPIC_NAME") => {
  const response = await snsClient.send(
    new CreateTopicCommand({ Name: topicName }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '087b8ad2-4593-50c4-a496-d7e90b82cf3e',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   TopicArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:TOPIC_NAME'
  // }
  return response;
};
```

要运行示例，请在命令提示符中键入以下内容。

```
node create-topic.js
```

此示例代码可在 GitHub 上的此处找到。

列出主题

在本示例中，使用 Node.js 模块列出所有 Amazon SNS 主题。

创建一个 `libs` 目录，然后使用文件名 `snsClient.js` 创建一个 Node.js 模块。将以下代码复制并粘贴到其中，这将创建 Amazon SNS 客户端对象。将 *REGION* 替换为您的 Amazon 区域。

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

此示例代码可在 GitHub 上的此处找到。

创建文件名为 `list-topics.js` 的 Node.js 模块。如前所示配置 SDK，包括安装所需的客户端和软件包。

创建一个空对象以传递到 SNS 客户端类的 `ListTopicsCommand` 方法。要调用 `ListTopicsCommand` 方法，请创建一个用于调用 Amazon SNS 服务对象的异步函数并传递参数对象。返回的 `data` 包含您的主题 Amazon 资源名称 (ARN) 的一个数组。

```
import { ListTopicsCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

export const listTopics = async () => {
  const response = await snsClient.send(new ListTopicsCommand({}));
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '936bc5ad-83ca-53c2-b0b7-9891167b909e',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   Topics: [ { TopicArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:mytopic' } ]
  // }
  return response;
};
```

要运行示例，请在命令提示符中键入以下内容。

```
node list-topics.js
```

此示例代码可在 GitHub 上的此处找到。

删除主题

在本示例中，使用 Node.js 模块删除 Amazon SNS 主题。

创建一个 `libs` 目录，然后使用文件名 `snsClient.js` 创建一个 Node.js 模块。将以下代码复制并粘贴到其中，这将创建 Amazon SNS 客户端对象。将 *REGION* 替换为您的 Amazon 区域。

```
import { SNSClient } from "@aws-sdk/client-sns";
```

```
// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

此示例代码可在 GitHub 上的此处找到。

创建文件名为 `delete-topic.js` 的 Node.js 模块。如前所示配置 SDK，包括安装所需的客户端和软件包。

创建包含要删除的主题的 `TopicArn` 的对象，将其传递到 SNS 客户端类的 `DeleteTopicCommand` 方法。要调用 `DeleteTopicCommand`方法，请创建一个异步函数，调用 Amazon SNS 客户端服务对象并传递参数对象。

> **Note**
>
> 将 *TOPIC_ARN* 替换为要删除的主题的 Amazon 资源名称 (ARN)。

```
import { DeleteTopicCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic to delete.
 */
export const deleteTopic = async (topicArn = "TOPIC_ARN") => {
  const response = await snsClient.send(
    new DeleteTopicCommand({ TopicArn: topicArn }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'a10e2886-5a8f-5114-af36-75bd39498332',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }
};
```

要运行示例，请在命令提示符中键入以下内容。

```
node delete-topic.js
```

此示例代码可在 GitHub 上的此处找到。

获取主题属性

在本示例中，使用 Node.js 模块检索 Amazon SNS 主题的属性。

创建一个 `libs` 目录，然后使用文件名 `snsClient.js` 创建一个 Node.js 模块。将以下代码复制并粘贴到其中，这将创建 Amazon SNS 客户端对象。将 *REGION* 替换为您的 Amazon 区域。

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

此示例代码可在 GitHub 上的此处找到。

创建文件名为 `get-topic-attributes.js` 的 Node.js 模块。按前面所示配置 SDK。

创建包含要删除主题的 `TopicArn` 的对象，将其传递到 SNS 客户端类的 `GetTopicAttributesCommand` 方法。要调用 `GetTopicAttributesCommand` 方法，请调用一个 Amazon SNS 客户端服务对象来传递参数对象。

> **Note**
>
> 将 *TOPIC_ARN* 替换为主题的 ARN。

```
import { GetTopicAttributesCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic to retrieve attributes for.
 */
export const getTopicAttributes = async (topicArn = "TOPIC_ARN") => {
  const response = await snsClient.send(
```

```
    new GetTopicAttributesCommand({
      TopicArn: topicArn,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '36b6a24e-5473-5d4e-ac32-ff72d9a73d94',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   Attributes: {
  //     Policy: '{...}',
  //     Owner: 'xxxxxxxxxxxx',
  //     SubscriptionsPending: '1',
  //     TopicArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:mytopic',
  //     TracingConfig: 'PassThrough',
  //     EffectiveDeliveryPolicy: '{"http":{"defaultHealthyRetryPolicy":
{"minDelayTarget":20,"maxDelayTarget":20,"numRetries":3,"numMaxDelayRetries":0,"numNoDelayRetri
{"headerContentType":"text/plain; charset=UTF-8"}}}',
  //     SubscriptionsConfirmed: '0',
  //     DisplayName: '',
  //     SubscriptionsDeleted: '1'
  //   }
  // }
  return response;
};
```

要运行示例，请在命令提示符中键入以下内容。

```
node get-topic-attributes.js
```

此示例代码可在 GitHub 上的此处找到。

设置主题属性

在本示例中，使用 Node.js 模块设置 Amazon SNS 主题的可变属性。

创建一个 `libs` 目录，然后使用文件名 `snsClient.js` 创建一个 Node.js 模块。将以下代码复制并粘贴到其中，这将创建 Amazon SNS 客户端对象。将 *REGION* 替换为您的 Amazon 区域。

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

此示例代码可在 GitHub 上的此处找到。

创建文件名为 `set-topic-attributes.js` 的 Node.js 模块。按前面所示配置 SDK。

创建包含用于属性更新参数的对象，这包括要设置其属性的主题的 `TopicArn`、要设置的属性的名称以及该属性的新值。您只能设置 `Policy`、`DisplayName` 和 `DeliveryPolicy` 属性。将参数传递到 SNS 客户端类的 `SetTopicAttributesCommand` 方法。要调用 `SetTopicAttributesCommand`方法，请创建一个异步函数，调用 Amazon SNS 客户端服务对象并传递参数对象。

> **Note**
>
> 将 *ATTRIBUTE_NAME* 替换为您正在设置的属性的名称，将 *TOPIC_ARN* 替换为您要设置属性的主题的 Amazon 资源名称 (ARN)，将 *NEW_ATTRIBUTE_VALUE* 替换为该属性的新值。

```
import { SetTopicAttributesCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

export const setTopicAttributes = async (
  topicArn = "TOPIC_ARN",
  attributeName = "DisplayName",
  attributeValue = "Test Topic",
) => {
  const response = await snsClient.send(
    new SetTopicAttributesCommand({
      AttributeName: attributeName,
      AttributeValue: attributeValue,
      TopicArn: topicArn,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
```

```
  //      requestId: 'd1b08d0e-e9a4-54c3-b8b1-d03238d2b935',
  //      extendedRequestId: undefined,
  //      cfId: undefined,
  //      attempts: 1,
  //      totalRetryDelay: 0
  //    }
  // }
  return response;
};
```

要运行示例，请在命令提示符中键入以下内容。

```
node set-topic-attributes.js
```

此示例代码可在 GitHub 上的此处找到。

在 Amazon SNS 中发布消息

此 Node.js 代码示例演示：

- 如何将消息发布到 Amazon SNS 主题。

情景

在本示例中，您使用一系列 Node.js 模块，将消息从 Amazon SNS 发布到主题端点、电子邮件或电话号码。Node.js 模块使用 SDK for JavaScript，通过 SNS 客户端类的以下方法发送消息：

- `PublishCommand`

先决条件任务

要设置和运行此示例，您必须先完成以下任务：

- 设置项目环境以运行这些 Node TypeScript 示例，并安装所需的 Amazon SDK for JavaScript 和第三方模块。请按照 GitHub 上的说明进行操作。

- 使用用户凭证创建共享配置文件。有关提供共享凭证文件的更多信息，请参阅《Amazon SDK 和工具参考指南》 中的共享配置和凭证文件。

> **Important**
>
> 这些示例演示了如何使用 ECMAScript6 (ES6) 导入/导出客户端服务对象和命令。
>
> - 这需要使用 Node.js 版本 13.x 或更高版本。要下载并安装最新版本的 Node.js，请参阅 Node.js 下载。
> - 如果您更喜欢使用 CommonJS 语法，请参阅 JavaScript ES6/CommonJS 语法。

将消息发布到 SNS 主题

在本示例中，使用 Node.js 模块将消息发布到 Amazon SNS 主题。

创建一个 `libs` 目录，然后使用文件名 `snsClient.js` 创建一个 Node.js 模块。将以下代码复制并粘贴到其中，这将创建 Amazon SNS 客户端对象。将 *REGION* 替换为您的 Amazon 区域。

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

此示例代码可在 GitHub 上的此处找到。

创建文件名为 `publish-topic.js` 的 Node.js 模块。按前面所示配置 SDK。

创建包含用于发布消息的参数的对象，包括消息文本以及 Amazon SNS 主题的 Amazon 资源名称 (ARN)。有关可用 SMS 属性的详细信息，请参阅SetSMSAttributes。

将参数传递到 SNS 客户端类的 `PublishCommand` 方法。创建一个异步函数，调用 Amazon SNS 客户端服务对象并传递参数对象。

> **Note**
>
> 将 *MESSAGE_TEXT* 替换为消息文本，将 *TOPIC_ARN* 替换为 SNS 主题的 ARN。

```
import { PublishCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string | Record<string, any>} message - The message to send. Can be a plain
 string or an object
 *                                                 if you are using the `json`
 `MessageStructure`.
 * @param {string} topicArn - The ARN of the topic to which you would like to publish.
 */
export const publish = async (
  message = "Hello from SNS!",
  topicArn = "TOPIC_ARN",
) => {
  const response = await snsClient.send(
    new PublishCommand({
      Message: message,
      TopicArn: topicArn,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'e7f77526-e295-5325-9ee4-281a43ad1f05',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   MessageId: 'xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx'
  // }
  return response;
};
```

要运行示例，请在命令提示符中键入以下内容。

```
node publish-topic.js
```

此示例代码可在 GitHub 上的此处找到。

在 Amazon SNS 中管理订阅

此 Node.js 代码示例演示：

- 如何列出对 Amazon SNS 主题的所有订阅。
- 如何将电子邮件地址、应用程序端点或 Amazon Lambda 函数订阅到 Amazon SNS 主题。
- 如何从 Amazon SNS 主题取消订阅。

情景

在本示例中，您使用一系列 Node.js 模块将通知消息发布到 Amazon SNS 主题。Node.js 模块使用 SDK for JavaScript，通过 SNS 客户端类的以下方法管理主题：

- `ListSubscriptionsByTopicCommand`
- `SubscribeCommand`
- `ConfirmSubscriptionCommand`
- `UnsubscribeCommand`

先决条件任务

要设置和运行此示例，您必须先完成以下任务：

- 设置项目环境以运行这些 Node TypeScript 示例，并安装所需的 Amazon SDK for JavaScript 和第三方模块。请按照 GitHub 上的说明进行操作。
- 使用用户凭证创建共享配置文件。有关提供共享凭证文件的更多信息，请参阅《Amazon SDK 和工具参考指南》 中的共享配置和凭证文件。

> **Important**
>
> 这些示例演示了如何使用 ECMAScript6 (ES6) 导入/导出客户端服务对象和命令。
>
> - 这需要使用 Node.js 版本 13.x 或更高版本。要下载并安装最新版本的 Node.js，请参阅 Node.js 下载。

- 如果您更喜欢使用 CommonJS 语法，请参阅 JavaScript ES6/CommonJS 语法。

列出对主题的订阅

在本示例中，使用 Node.js 模块以列出对 Amazon SNS 主题的所有订阅。

创建一个 `libs` 目录，然后使用文件名 `snsClient.js` 创建一个 Node.js 模块。将以下代码复制并粘贴到其中，这将创建 Amazon SNS 客户端对象。将 *REGION* 替换为您的 Amazon 区域。

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

此示例代码可在 GitHub 上的此处找到。

创建文件名为 `list-subscriptions-by-topic.js` 的 Node.js 模块。按前面所示配置 SDK。

创建一个对象，其中包含您要列出其订阅的主题的 `TopicArn` 参数。将参数传递到 SNS 客户端类的 `ListSubscriptionsByTopicCommand` 方法。要调用 `ListSubscriptionsByTopicCommand`方法，请创建一个异步函数，调用 Amazon SNS 客户端服务对象并传递参数对象。

> **Note**
>
> 将 *TOPIC_ARN* 替换为您要列出其订阅的主题的 Amazon 资源名称 (ARN)。

```
import { ListSubscriptionsByTopicCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic for which you wish to list
 subscriptions.
 */
export const listSubscriptionsByTopic = async (topicArn = "TOPIC_ARN") => {
  const response = await snsClient.send(
    new ListSubscriptionsByTopicCommand({ TopicArn: topicArn }),
  );
```

```
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '0934fedf-0c4b-572e-9ed2-a3e38fadb0c8',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   Subscriptions: [
  //     {
  //       SubscriptionArn: 'PendingConfirmation',
  //       Owner: '901487484989',
  //       Protocol: 'email',
  //       Endpoint: 'corepyle@amazon.com',
  //       TopicArn: 'arn:aws:sns:us-east-1:901487484989:mytopic'
  //     }
  //   ]
  // }
  return response;
};
```

要运行示例，请在命令提示符中键入以下内容。

```
node list-subscriptions-by-topic.js
```

此示例代码可在 GitHub 上的此处找到。

将电子邮件地址订阅到主题

在本示例中，使用 Node.js 模块来订阅电子邮件地址，使其从 Amazon SNS 主题接收 SMTP 电子邮件。

创建一个 `libs` 目录，然后使用文件名 `snsClient.js` 创建一个 Node.js 模块。将以下代码复制并粘贴到其中，这将创建 Amazon SNS 客户端对象。将 *REGION* 替换为您的 Amazon 区域。

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
```

```
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

此示例代码可在 GitHub 上的此处找到。

创建文件名为 subscribe-email.js 的 Node.js 模块。按前面所示配置 SDK。

创建包含 Protocol 参数的对象，用于指定 email 协议、要订阅到的主题的 TopicArn 以及作为邮件 Endpoint 的电子邮件地址。将参数传递到 SNS 客户端类的 SubscribeCommand 方法。您可以使用 subscribe 方法，根据在所传递参数中使用的值，将多种不同的端点订阅到某个 Amazon SNS 主题，如本主题中的其他示例所示。

要调用 SubscribeCommand方法，请创建一个异步函数，调用 Amazon SNS 客户端服务对象并传递参数对象。

Note

将 *TOPIC_ARN* 替换为该主题的 Amazon 资源名称 (ARN)，将 *EMAIL_ADDRESS* 替换为用于订阅的电子邮件地址。

```
import { SubscribeCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic for which you wish to confirm a
 subscription.
 * @param {string} emailAddress - The email address that is subscribed to the topic.
 */
export const subscribeEmail = async (
  topicArn = "TOPIC_ARN",
  emailAddress = "usern@me.com",
) => {
  const response = await snsClient.send(
    new SubscribeCommand({
      Protocol: "email",
      TopicArn: topicArn,
      Endpoint: emailAddress,
    }),
  );
```

```
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'c8e35bcd-b3c0-5940-9f66-06f6fcc108f0',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   SubscriptionArn: 'pending confirmation'
  // }
};
```

要运行示例，请在命令提示符中键入以下内容。

```
node subscribe-email.js
```

此示例代码可在 GitHub 上的此处找到。

确认订阅

在本示例中，使用 Node.js 模块，通过验证之前的 SUBSCRIBE 操作发送到端点的令牌来验证端点所有者接收消息的意图。

创建一个 `libs` 目录，然后使用文件名 `snsClient.js` 创建一个 Node.js 模块。将以下代码复制并粘贴到其中，这将创建 Amazon SNS 客户端对象。将 *REGION* 替换为您的 Amazon 区域。

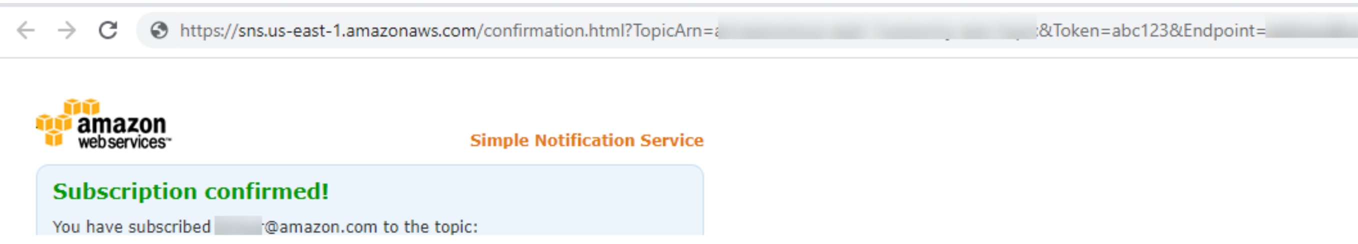
```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

此示例代码可在 GitHub 上的此处找到。

创建文件名为 `confirm-subscription.js` 的 Node.js 模块。如前所示配置 SDK，包括安装所需的客户端和软件包。

定义参数，包括 `TOPIC_ARN` 和 `TOKEN`，然后为 `AuthenticateOnUnsubscribe` 定义值 `TRUE` 或 `FALSE`。

令牌是在之前的 SUBSCRIBE 操作中发送给端点所有者的短期令牌。例如，对于电子邮件端点，TOKEN 可在发送给电子邮件所有者的确认订阅电子邮件的 URL 中找到。例如，在以下 URL 中，abc123 是令牌。



要调用 ConfirmSubscriptionCommand方法，请创建一个异步函数，调用 Amazon SNS 客户端服务对象并传递参数对象。

> **Note**
>
> 将 *TOPIC_ARN* 替换为主题的 Amazon 资源名称 (ARN)，将 *TOKEN* 替换为之前 Subscribe 操作中发送给端点所有者的 URL 中的令牌值，然后将 *AuthenticateOnUnsubscribe* 定义为值 TRUE 或 FALSE。

```
import { ConfirmSubscriptionCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} token - This token is sent the subscriber. Only subscribers
 *                         that are not AWS services (HTTP/S, email) need to be
 confirmed.
 * @param {string} topicArn - The ARN of the topic for which you wish to confirm a
 subscription.
 */
export const confirmSubscription = async (
  token = "TOKEN",
  topicArn = "TOPIC_ARN",
) => {
  const response = await snsClient.send(
    // A subscription only needs to be confirmed if the endpoint type is
    // HTTP/S, email, or in another AWS account.
    new ConfirmSubscriptionCommand({
      Token: token,
      TopicArn: topicArn,
      // If this is true, the subscriber cannot unsubscribe while unauthenticated.
```

```
      AuthenticateOnUnsubscribe: "false",
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '4bb5bce9-805a-5517-8333-e1d2cface90b',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   SubscriptionArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:TOPIC_NAME:xxxxxxxx-xxxx-
xxxx-xxxx-xxxxxxxxxxxx'
  // }
  return response;
};
```

要运行示例，请在命令提示符中键入以下内容。

```
node confirm-subscription.js
```

此示例代码可在 GitHub 上的此处找到。

将应用程序端点订阅到主题

在本示例中，使用 Node.js 模块来订阅移动应用程序端点，使其从 Amazon SNS 主题接收通知。

创建一个 `libs` 目录，然后使用文件名 `snsClient.js` 创建一个 Node.js 模块。将以下代码复制并粘贴到其中，这将创建 Amazon SNS 客户端对象。将 *REGION* 替换为您的 Amazon 区域。

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

此示例代码可在 GitHub 上的此处找到。

创建文件名为 `subscribe-app.js` 的 Node.js 模块。如前所示配置 SDK，包括安装所需的模块和软件包。

创建一个包含 `Protocol` 参数的对象，用于指定 `application` 协议、要订阅到的主题的 `TopicArn` 以及 `Endpoint` 参数的移动应用程序端点的 Amazon 资源名称 (ARN)。将参数传递到 SNS 客户端类的 `SubscribeCommand` 方法。

要调用 `SubscribeCommand` 方法，请创建一个用于调用 Amazon SNS 服务对象的异步函数并传递参数对象。

> **Note**
>
> 将 *TOPIC_ARN* 替换为主题的 Amazon 资源名称 (ARN)，将 *MOBILE_ENDPOINT_ARN* 替换为您订阅主题的端点。

```
import { SubscribeCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic the subscriber is subscribing to.
 * @param {string} endpoint - The Endpoint ARN of an application. This endpoint is
 created
 *                            when an application registers for notifications.
 */
export const subscribeApp = async (
  topicArn = "TOPIC_ARN",
  endpoint = "ENDPOINT",
) => {
  const response = await snsClient.send(
    new SubscribeCommand({
      Protocol: "application",
      TopicArn: topicArn,
      Endpoint: endpoint,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'c8e35bcd-b3c0-5940-9f66-06f6fcc108f0',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
```

```
  //   },
  //   SubscriptionArn: 'pending confirmation'
  // }
  return response;
};
```

要运行示例，请在命令提示符中键入以下内容。

```
node subscribe-app.js
```

此示例代码可在 GitHub 上的此处找到。

将 Lambda 函数订阅到主题

在本示例中，使用 Node.js 模块来订阅 Amazon Lambda 函数，使其从 Amazon SNS 主题接收通知。

创建一个 `libs` 目录，然后使用文件名 `snsClient.js` 创建一个 Node.js 模块。将以下代码复制并粘贴到其中，这将创建 Amazon SNS 客户端对象。将 *REGION* 替换为您的 Amazon 区域。

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

此示例代码可在 GitHub 上的此处找到。

创建文件名为 `subscribe-lambda.js` 的 Node.js 模块。按前面所示配置 SDK。

创建一个包含 `Protocol` 参数的对象，指定 `lambda` 协议、要订阅到的主题的 `TopicArn` 以及作为 `Endpoint` 参数的 Amazon Lambda 函数的 Amazon 资源名称 (ARN)。将参数传递到 SNS 客户端类的 `SubscribeCommand` 方法。

要调用 `SubscribeCommand`方法，请创建一个异步函数，调用 Amazon SNS 客户端服务对象并传递参数对象。

> **Note**
>
> 将 *TOPIC_ARN* 替换为主题的 Amazon 资源名称 (ARN)，将 *LAMBDA_FUNCTION_ARN* 替换为 Lambda 函数的 Amazon 资源名称 (ARN)。

```
import { SubscribeCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic the subscriber is subscribing to.
 * @param {string} endpoint - The Endpoint ARN of and AWS Lambda function.
 */
export const subscribeLambda = async (
  topicArn = "TOPIC_ARN",
  endpoint = "ENDPOINT",
) => {
  const response = await snsClient.send(
    new SubscribeCommand({
      Protocol: "lambda",
      TopicArn: topicArn,
      Endpoint: endpoint,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'c8e35bcd-b3c0-5940-9f66-06f6fcc108f0',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   SubscriptionArn: 'pending confirmation'
  // }
  return response;
};
```

要运行示例，请在命令提示符中键入以下内容。

```
node subscribe-lambda.js
```

此示例代码可在 GitHub 上的此处找到。

从主题取消订阅

在本示例中，使用 Node.js 模块取消订阅 Amazon SNS 主题订阅。

创建一个 libs 目录，然后使用文件名 snsClient.js 创建一个 Node.js 模块。将以下代码复制并粘贴到其中，这将创建 Amazon SNS 客户端对象。将 *REGION* 替换为您的 Amazon 区域。

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

此示例代码可在 GitHub 上的此处找到。

创建文件名为 unsubscribe.js 的 Node.js 模块。如前所示配置 SDK，包括安装所需的客户端和软件包。

创建一个包含 SubscriptionArn 参数的对象，指定要取消订阅的订阅的 Amazon 资源名称 (ARN)。将参数传递到 SNS 客户端类的 UnsubscribeCommand 方法。

要调用 UnsubscribeCommand方法，请创建一个异步函数，调用 Amazon SNS 客户端服务对象并传递参数对象。

> **Note**
>
> 将 *TOPIC_SUBSCRIPTION_ARN* 替换为要取消订阅的订阅的 Amazon 资源名称（ARN）。

```
import { UnsubscribeCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} subscriptionArn - The ARN of the subscription to cancel.
 */
const unsubscribe = async (
  subscriptionArn = "arn:aws:sns:us-east-1:xxxxxxxxxxxx:mytopic:xxxxxxxx-xxxx-xxxx-
xxxx-xxxxxxxxxxxx",
) => {
  const response = await snsClient.send(
    new UnsubscribeCommand({
      SubscriptionArn: subscriptionArn,
    }),
  );
```

```
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '0178259a-9204-507c-b620-78a7570a44c6',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }
  return response;
};
```

要运行示例，请在命令提示符中键入以下内容。

```
node unsubscribe.js
```

此示例代码可在 GitHub 上的此处找到。

使用 Amazon SNS 发送 SMS 消息

此 Node.js 代码示例演示：

- 如何获取和设置 Amazon SNS 的 SMS 消息发送首选项。
- 如何检查电话号码以确定是否选择退出接收 SMS 消息。
- 如何获取已选择退出接收 SMS 消息的电话号码列表。
- 如何发送 SMS 消息。

情景

您可以使用 Amazon SNS 将文本消息或 SMS 消息发送到支持 SMS 的设备上。您可以直接向电话号码发送消息，也可以使用多个电话号码订阅主题，然后通过向该主题发送消息来一次向这些电话号码发送消息。

在本示例中，您使用一系列 Node.js 模块将 SMS 文本消息从 Amazon SNS 发送到支持 SMS 的设备。Node.js 模块使用 SDK for JavaScript，通过 SNS 客户端类的以下方法发布 SMS 消息：

- `GetSMSAttributesCommand`
- `SetSMSAttributesCommand`
- `CheckIfPhoneNumberIsOptedOutCommand`
- `ListPhoneNumbersOptedOutCommand`
- `PublishCommand`

先决条件任务

要设置和运行此示例，您必须先完成以下任务：

- 设置项目环境以运行这些 Node TypeScript 示例，并安装所需的 Amazon SDK for JavaScript 和第三方模块。请按照 GitHub 上的说明进行操作。
- 使用用户凭证创建共享配置文件。有关提供共享凭证文件的更多信息，请参阅《Amazon SDK 和工具参考指南》 中的共享配置和凭证文件。

> **Important**
>
> 这些示例演示了如何使用 ECMAScript6 (ES6) 导入/导出客户端服务对象和命令。
>
> - 这需要使用 Node.js 版本 13.x 或更高版本。要下载并安装最新版本的 Node.js，请参阅 Node.js 下载。
> - 如果您更喜欢使用 CommonJS 语法，请参阅 JavaScript ES6/CommonJS 语法。

获取 SMS 属性

使用 Amazon SNS 来指定发送 SMS 消息的首选项，例如如何优化消息传输（在成本或可靠传输方面）、您的每月支出限额、如何记录消息传输以及是否要订阅每日 SMS 使用率报告。这些首选项通过检索得到，并设置为 Amazon SNS 的 SMS 属性。

在本示例中，使用 Node.js 模块获取 Amazon SNS 中的当前 SMS 属性。

创建一个 `libs` 目录，然后使用文件名 `snsClient.js` 创建一个 Node.js 模块。将以下代码复制并粘贴到其中，这将创建 Amazon SNS 客户端对象。将 *`REGION`* 替换为您的 Amazon 区域。

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

此示例代码可在 GitHub 上的此处找到。

创建文件名为 `get-sms-attributes.js` 的 Node.js 模块。

如前所示配置 SDK，包括下载所需的客户端和软件包。创建包含用于获取 SMS 属性的参数的对象，包括要获取的单个属性的名称。有关可用 SMS 属性的详细信息，请参阅《Amazon Simple Notification Service API 参考》中的 SetSMSAttributes。

此示例获取 `DefaultSMSType` 属性，该属性控制 SMS 消息是作为 `Promotional` 发送（这将优化消息传送以尽可能降低成本）还是作为 `Transactional` 发送（这将优化消息传送以实现最高的可靠性）。将参数传递到 SNS 客户端类的 `SetTopicAttributesCommand` 方法。要调用 `SetSMSAttributesCommand`方法，请创建一个异步函数，调用 Amazon SNS 客户端服务对象并传递参数对象。

> **Note**
>
> 将 *ATTRIBUTE_NAME* 替换为属性的名称。

```
import { GetSMSAttributesCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

export const getSmsAttributes = async () => {
  const response = await snsClient.send(
    // If you have not modified the account-level mobile settings of SNS,
    // the DefaultSMSType is undefined. For this example, it was set to
    // Transactional.
    new GetSMSAttributesCommand({ attributes: ["DefaultSMSType"] }),
  );

  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
```

```
  //     requestId: '67ad8386-4169-58f1-bdb9-debd281d48d5',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   attributes: { DefaultSMSType: 'Transactional' }
  // }
  return response;
};
```

要运行示例，请在命令提示符中键入以下内容。

```
node get-sms-attributes.js
```

此示例代码可在 GitHub 上的此处找到。

设置 SMS 属性

在本示例中，使用 Node.js 模块获取 Amazon SNS 中的当前 SMS 属性。

创建一个 `libs` 目录，然后使用文件名 `snsClient.js` 创建一个 Node.js 模块。将以下代码复制并粘贴到其中，这将创建 Amazon SNS 客户端对象。将 *REGION* 替换为您的 Amazon 区域。

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

此示例代码可在 GitHub 上的此处找到。

创建文件名为 `set-sms-attribute-type.js` 的 Node.js 模块。如前所示配置 SDK，包括安装所需的客户端和软件包。创建包含用于设置 SMS 属性的参数的对象，其中包括要设置的单个属性的名称以及为各个属性设置的值。有关可用 SMS 属性的详细信息，请参阅《Amazon Simple Notification Service API 参考》中的 SetSMSAttributes。

此示例将 `DefaultSMSType` 属性设置为 `Transactional`，这会优化消息传送以实现最高的可靠性。将参数传递到 SNS 客户端类的 `SetTopicAttributesCommand` 方法。要调用 `SetSMSAttributesCommand`方法，请创建一个异步函数，调用 Amazon SNS 客户端服务对象并传递参数对象。

```
import { SetSMSAttributesCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {"Transactional" | "Promotional"} defaultSmsType
 */
export const setSmsType = async (defaultSmsType = "Transactional") => {
  const response = await snsClient.send(
    new SetSMSAttributesCommand({
      attributes: {
        // Promotional – (Default) Noncritical messages, such as marketing messages.
        // Transactional – Critical messages that support customer transactions,
        // such as one-time passcodes for multi-factor authentication.
        DefaultSMSType: defaultSmsType,
      },
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '1885b977-2d7e-535e-8214-e44be727e265',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }
  return response;
};
```

要运行示例，请在命令提示符中键入以下内容。

```
node set-sms-attribute-type.js
```

此示例代码可在 GitHub 上的此处找到。

检查电话号码是否已选择不接收消息

在本示例中，使用 Node.js 模块检查电话号码，确定该号码是否已退出接收 SMS 消息。

创建一个 `libs` 目录，然后使用文件名 `snsClient.js` 创建一个 Node.js 模块。将以下代码复制并粘贴到其中，这将创建 Amazon SNS 客户端对象。将 *REGION* 替换为您的 Amazon 区域。

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

此示例代码可在 GitHub 上的此处找到。

创建文件名为 `check-if-phone-number-is-opted-out.js` 的 Node.js 模块。按前面所示配置 SDK。创建一个对象，其中将要检查的电话号码包含作为参数。

此示例设置 `PhoneNumber` 参数以指定要检查的电话号码。将对象发布到 SNS 客户端类的 `CheckIfPhoneNumberIsOptedOutCommand` 方法。要调用 `CheckIfPhoneNumberIsOptedOutCommand`方法，请创建一个异步函数，调用 Amazon SNS 客户端服务对象并传递参数对象。

> **Note**
>
> 1.
>
>   将 *PHONE_NUMBER* 替换为电话号码。

```
import { CheckIfPhoneNumberIsOptedOutCommand } from "@aws-sdk/client-sns";

import { snsClient } from "../libs/snsClient.js";

export const checkIfPhoneNumberIsOptedOut = async (
  phoneNumber = "5555555555",
) => {
  const command = new CheckIfPhoneNumberIsOptedOutCommand({
    phoneNumber,
  });

  const response = await snsClient.send(command);
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
```

```
  //     requestId: '3341c28a-cdc8-5b39-a3ee-9fb0ee125732',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   isOptedOut: false
  // }
  return response;
};
```

要运行示例，请在命令提示符中键入以下内容。

```
node check-if-phone-number-is-opted-out.js
```

此示例代码可在 GitHub 上的此处找到。

列出已退出的电话号码

在本示例中，使用 Node.js 模块获取已退出接收 SMS 消息的电话号码列表。

创建一个 `libs` 目录，然后使用文件名 `snsClient.js` 创建一个 Node.js 模块。将以下代码复制并粘贴到其中，这将创建 Amazon SNS 客户端对象。将 *REGION* 替换为您的 Amazon 区域。

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

此示例代码可在 GitHub 上的此处找到。

创建文件名为 `list-phone-numbers-opted-out.js` 的 Node.js 模块。按前面所示配置 SDK。创建一个空对象作为参数。

将对象发布到 SNS 客户端类的 `ListPhoneNumbersOptedOutCommand` 方法。要调用 `ListPhoneNumbersOptedOutCommand`方法，请创建一个异步函数，调用 Amazon SNS 客户端服务对象并传递参数对象。

```
import { ListPhoneNumbersOptedOutCommand } from "@aws-sdk/client-sns";
```

```
import { snsClient } from "../libs/snsClient.js";

export const listPhoneNumbersOptedOut = async () => {
  const response = await snsClient.send(
    new ListPhoneNumbersOptedOutCommand({}),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '44ff72fd-1037-5042-ad96-2fc16601df42',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   phoneNumbers: ['+15555550100']
  // }
  return response;
};
```

要运行示例，请在命令提示符中键入以下内容。

```
node list-phone-numbers-opted-out.js
```

此示例代码可在 GitHub 上的此处找到。

发布 SMS 消息

在本示例中，使用 Node.js 模块将 SMS 消息发布到电话号码。

创建一个 `libs` 目录，然后使用文件名 `snsClient.js` 创建一个 Node.js 模块。将以下代码复制并粘贴到其中，这将创建 Amazon SNS 客户端对象。将 *REGION* 替换为您的 Amazon 区域。

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

此示例代码可在 GitHub 上的此处找到。

创建文件名为 publish-sms.js 的 Node.js 模块。如前所示配置 SDK，包括安装所需的客户端和软件包。创建一个包含 Message 和 PhoneNumber 参数的对象。

在发送 SMS 消息时，请使用 E.164 格式指定电话号码。E.164 是用于国际电信的电话号码结构标准。遵循此格式的电话号码最多可包含 15 位，并以加号 (+) 和国家/地区代码作为前缀。例如，E.164 格式的美国电话号码将显示为 +1001XXX5550100。

此示例设置 PhoneNumber 参数以指定将消息发送到的电话号码。将对象发布到 SNS 客户端类的 PublishCommand 方法。要调用 PublishCommand 方法，请创建一个用于调用 Amazon SNS 服务对象的异步函数并传递参数对象。

> **Note**
>
> 将 *TEXT_MESSAGE* 替换为文本消息，将 *PHONE_NUMBER* 替换为电话号码。

```
import { PublishCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string | Record<string, any>} message - The message to send. Can be a plain
 string or an object
 *                                                 if you are using the `json`
 `MessageStructure`.
 * @param {*} phoneNumber - The phone number to send the message to.
 */
export const publish = async (
  message = "Hello from SNS!",
  phoneNumber = "+15555555555",
) => {
  const response = await snsClient.send(
    new PublishCommand({
      Message: message,
      // One of PhoneNumber, TopicArn, or TargetArn must be specified.
      PhoneNumber: phoneNumber,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '7410094f-efc7-5f52-af03-54737569ab77',
```

```
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   MessageId: 'xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx'
  // }
  return response;
};
```

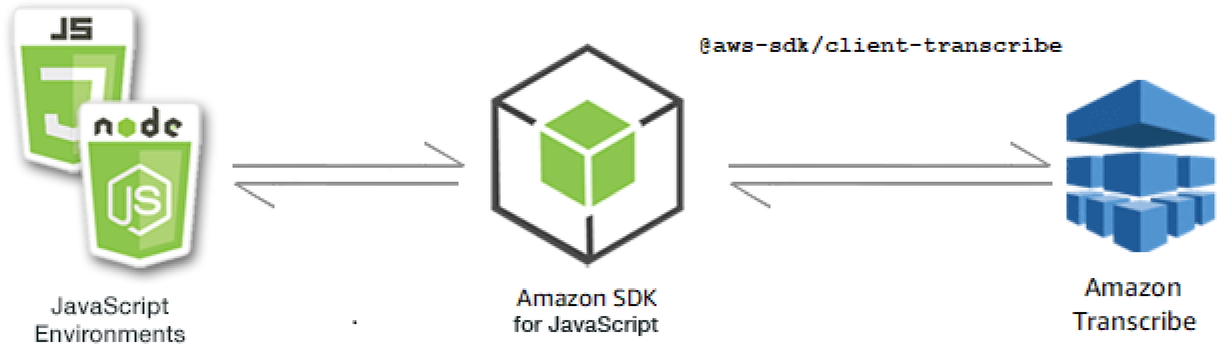
要运行示例，请在命令提示符中键入以下内容。

```
node publish-sms.js
```

此示例代码可在 GitHub 上的此处找到。

Amazon Transcribe 示例

Amazon Transcribe 使开发人员能够轻松地向其应用程序添加语音到文本转换功能。



适用于 Amazon Transcribe 的 JavaScript API 通过 TranscribeService 客户端类公开。

主题

- Amazon Transcribe 示例
- Amazon Transcribe Medical 示例

Amazon Transcribe 示例

在此示例中，使用一系列 Node.js 模块通过 `TranscribeService` 客户端类的以下方法创建、列出和删除转录作业：

- `StartTranscriptionJobCommand`
- `ListTranscriptionJobsCommand`
- `DeleteTranscriptionJobCommand`

有关 Amazon Transcribe 的更多信息，请参阅 Amazon Transcribe 开发人员指南。

先决条件任务

要设置和运行此示例，您必须先完成以下任务：

- 设置项目环境以运行这些 Node TypeScript 示例，并安装所需的 Amazon SDK for JavaScript 和第三方模块。请按照 GitHub 上的说明进行操作。
- 使用用户凭证创建共享配置文件。有关提供共享凭证文件的更多信息，请参阅《Amazon SDK 和工具参考指南》 中的共享配置和凭证文件。

> **Important**
>
> 这些示例演示了如何使用 ECMAScript6 (ES6) 导入/导出客户端服务对象和命令。
>
> - 这需要使用 Node.js 版本 13.x 或更高版本。要下载并安装最新版本的 Node.js，请参阅 Node.js 下载。
> - 如果您更喜欢使用 CommonJS 语法，请参阅 JavaScript ES6/CommonJS 语法

启动 Amazon Transcribe 作业

此示例演示如何使用 Amazon SDK for JavaScript 启动 Amazon Transcribe 转录作业。有关更多信息，请参阅 StartTranscriptionJobCommand。

创建一个 `libs` 目录，然后使用文件名 `transcribeClient.js` 创建一个 Node.js 模块。将以下代码复制并粘贴到其中，这将创建 Amazon Transcribe 客户端对象。将 *REGION* 替换为您的 Amazon 区域。

```
const { TranscribeClient } = require("@aws-sdk/client-transcribe");
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon Transcribe service client object.
const transcribeClient = new TranscribeClient({ region: REGION });
```

```
export { transcribeClient };
```

此示例代码可在 GitHub 上的此处找到。

创建文件名为 `transcribe-create-job.js` 的 Node.js 模块。确保如前所示配置 SDK，包括安装所需的客户端和软件包。创建一个参数对象，指定所需的参数。使用 `StartMedicalTranscriptionJobCommand` 命令启动作业。

> **Note**
>
> 将 *MEDICAL_JOB_NAME* 替换为转录作业的名称。对于 *OUTPUT_BUCKET_NAME*，指定用于保存输出的 Amazon S3 存储桶。对于 *JOB_TYPE*，请指定作业类型。对于 *SOURCE_LOCATION*，指定源文件的位置。对于 *SOURCE_FILE_LOCATION*，指定输入媒体文件的位置。

```
// Import the required AWS SDK clients and commands for Node.js
import { StartTranscriptionJobCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "./libs/transcribeClient.js";

// Set the parameters
export const params = {
  TranscriptionJobName: "JOB_NAME",
  LanguageCode: "LANGUAGE_CODE", // For example, 'en-US'
  MediaFormat: "SOURCE_FILE_FORMAT", // For example, 'wav'
  Media: {
    MediaFileUri: "SOURCE_LOCATION",
    // For example, "https://transcribe-demo.s3-REGION.amazonaws.com/hello_world.wav"
  },
  OutputBucketName: "OUTPUT_BUCKET_NAME"
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new StartTranscriptionJobCommand(params)
    );
    console.log("Success - put", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
```

```
};
run();
```

要运行示例，请在命令提示符中键入以下内容。

```
node transcribe-create-job.js
```

此示例代码可在 GitHub 上的此处找到。

列出 Amazon Transcribe 作业

此示例演示如何使用 Amazon SDK for JavaScript 列出 Amazon Transcribe 转录作业。有关您可以修改的其他设置的更多信息，请参阅 `ListTranscriptionJobCommand`。

创建一个 `libs` 目录，然后使用文件名 `transcribeClient.js` 创建一个 Node.js 模块。将以下代码复制并粘贴到其中，这将创建 Amazon Transcribe 客户端对象。将 *`REGION`* 替换为您的 Amazon 区域。

```
const { TranscribeClient } = require("@aws-sdk/client-transcribe");
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon Transcribe service client object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

此示例代码可在 GitHub 上的此处找到。

创建文件名为 `transcribe-list-jobs.js` 的 Node.js 模块。确保如前所示配置 SDK，包括安装所需的客户端和软件包。使用所需参数创建参数对象。

> **Note**
>
> 将 *`KEY_WORD`* 替换为返回的作业名称必须包含的关键字。

```
// Import the required AWS SDK clients and commands for Node.js

import { ListTranscriptionJobsCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "./libs/transcribeClient.js";
```

```
// Set the parameters
export const params = {
  JobNameContains: "KEYWORD", // Not required. Returns only transcription
  // job names containing this string
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new ListTranscriptionJobsCommand(params)
    );
    console.log("Success", data.TranscriptionJobSummaries);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

要运行示例，请在命令提示符中键入以下内容。

```
node transcribe-list-jobs.js
```

此示例代码可在 GitHub 上的此处找到。

删除 Amazon Transcribe 作业

此示例演示如何使用 Amazon SDK for JavaScript 删除 Amazon Transcribe 转录作业。有关选项的更多信息，请参阅 `DeleteTranscriptionJobCommand`。

创建一个 `libs` 目录，然后使用文件名 `transcribeClient.js` 创建一个 Node.js 模块。将以下代码复制并粘贴到其中，这将创建 Amazon Transcribe 客户端对象。将 *REGION* 替换为您的 Amazon 区域。

```
import  { TranscribeClient }  from  "@aws-sdk/client-transcribe";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create Transcribe service object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

此示例代码可在 GitHub 上的此处找到。

创建文件名为 `transcribe-delete-job.js` 的 Node.js 模块。确保如前所示配置 SDK，包括安装所需的客户端和软件包。指定要删除的作业的 Amazon 区域和名称。

Note

将 *JOB_NAME* 替换为要删除的作业的名称。

```
// Import the required AWS SDK clients and commands for Node.js
import { DeleteTranscriptionJobCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "./libs/transcribeClient.js";

// Set the parameters
export const params = {
  TranscriptionJobName: "JOB_NAME", // Required. For example, 'transciption_demo'
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new DeleteTranscriptionJobCommand(params)
    );
    console.log("Success - deleted");
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

要运行示例，请在命令提示符中键入以下内容。

```
node transcribe-delete-job.js
```

此示例代码可在 GitHub 上的此处找到。

Amazon Transcribe Medical 示例

在此示例中，使用一系列 Node.js 模块通过 `TranscribeService` 客户端类的以下方法创建、列出和删除医疗转录作业：

- `StartMedicalTranscriptionJobCommand`
- `ListMedicalTranscriptionJobsCommand`
- `DeleteMedicalTranscriptionJobCommand`

有关 Amazon Transcribe 的更多信息，请参阅 Amazon Transcribe 开发人员指南。

先决条件任务

要设置和运行此示例，您必须先完成以下任务：

- 设置项目环境以运行这些 Node TypeScript 示例，并安装所需的 Amazon SDK for JavaScript 和第三方模块。请按照 GitHub 上的说明进行操作。
- 使用用户凭证创建共享配置文件。有关提供共享凭证文件的更多信息，请参阅《Amazon SDK 和工具参考指南》 中的共享配置和凭证文件。

> **Important**
>
> 这些示例演示了如何使用 ECMAScript6 (ES6) 导入/导出客户端服务对象和命令。
>
> - 这需要使用 Node.js 版本 13.x 或更高版本。要下载并安装最新版本的 Node.js，请参阅 Node.js 下载。
> - 如果您更喜欢使用 CommonJS 语法，请参阅 JavaScript ES6/CommonJS 语法

启动 Amazon Transcribe Medical 转录作业

此示例演示如何使用 Amazon SDK for JavaScript 启动 Amazon Transcribe Medical 转录作业。有关更多信息，请参阅 startMedicalTranscriptionJob。

创建一个 `libs` 目录，然后使用文件名 `transcribeClient.js` 创建一个 Node.js 模块。将以下代码复制并粘贴到其中，这将创建 Amazon Transcribe 客户端对象。将 *`REGION`* 替换为您的 Amazon 区域。

```
import  { TranscribeClient }  from  "@aws-sdk/client-transcribe";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create Transcribe service object.
const transcribeClient = new TranscribeClient({ region: REGION });
```

```
export { transcribeClient };
```

此示例代码可在 GitHub 上的此处找到。

创建文件名为 `transcribe-create-medical-job.js` 的 Node.js 模块。确保如前所示配置 SDK，包括安装所需的客户端和软件包。创建一个参数对象，指定所需的参数。使用 `StartMedicalTranscriptionJobCommand` 命令启动医疗作业。

> **Note**
>
> 将 *MEDICAL_JOB_NAME* 替换为医疗转录作业的名称。对于 *OUTPUT_BUCKET_NAME*，指定用于保存输出的 Amazon S3 存储桶。对于 *JOB_TYPE*，请指定作业类型。对于 *SOURCE_LOCATION*，指定源文件的位置。对于 *SOURCE_FILE_LOCATION*，指定输入媒体文件的位置。

```
// Import the required AWS SDK clients and commands for Node.js
import { StartMedicalTranscriptionJobCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "./libs/transcribeClient.js";

// Set the parameters
export const params = {
  MedicalTranscriptionJobName: "MEDICAL_JOB_NAME", // Required
  OutputBucketName: "OUTPUT_BUCKET_NAME", // Required
  Specialty: "PRIMARYCARE", // Required. Possible values are 'PRIMARYCARE'
  Type: "JOB_TYPE", // Required. Possible values are 'CONVERSATION' and 'DICTATION'
  LanguageCode: "LANGUAGE_CODE", // For example, 'en-US'
  MediaFormat: "SOURCE_FILE_FORMAT", // For example, 'wav'
  Media: {
    MediaFileUri: "SOURCE_FILE_LOCATION",
    // The S3 object location of the input media file. The URI must be in the same
 region
    // as the API endpoint that you are calling.For example,
    // "https://transcribe-demo.s3-REGION.amazonaws.com/hello_world.wav"
  },
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new StartMedicalTranscriptionJobCommand(params)
```

```
    );
    console.log("Success - put", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

要运行示例，请在命令提示符中键入以下内容。

```
node transcribe-create-medical-job.js
```

此示例代码可在 GitHub 上的此处找到。

列出 Amazon Transcribe Medical 作业

此示例演示如何使用 Amazon SDK for JavaScript 列出 Amazon Transcribe 转录作业。有关更多信息，请参阅 ListTranscriptionMedicalJobsCommand。

创建一个 `libs` 目录，然后使用文件名 `transcribeClient.js` 创建一个 Node.js 模块。将以下代码复制并粘贴到其中，这将创建 Amazon Transcribe 客户端对象。将 *REGION* 替换为您的 Amazon 区域。

```
const { TranscribeClient } = require("@aws-sdk/client-transcribe");
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon Transcribe service client object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

此示例代码可在 GitHub 上的此处找到。

创建文件名为 `transcribe-list-medical-jobs.js` 的 Node.js 模块。确保如前所示配置 SDK，包括安装所需的客户端和软件包。使用所需参数创建参数对象，并使用 `ListMedicalTranscriptionJobsCommand` 命令列出医疗作业。

> **Note**
>
> 将 *KEYWORD* 替换为返回的作业名称必须包含的关键字。

```
// Import the required AWS SDK clients and commands for Node.js

import { ListMedicalTranscriptionJobsCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "./libs/transcribeClient.js";

// Set the parameters
export const params = {
  JobNameContains: "KEYWORD", // Returns only transcription job names containing this
 string
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new ListMedicalTranscriptionJobsCommand(params)
    );
    console.log("Success", data.MedicalTranscriptionJobName);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

要运行示例，请在命令提示符中键入以下内容。

```
node transcribe-list-medical-jobs.js
```

此示例代码可在 GitHub 上的此处找到。

删除 Amazon Transcribe Medical 作业

此示例演示如何使用 Amazon SDK for JavaScript 删除 Amazon Transcribe 转录作业。有关选项的更多信息，请参阅 `DeleteTranscriptionMedicalJobCommand`。

创建一个 `libs` 目录，然后使用文件名 `transcribeClient.js` 创建一个 Node.js 模块。将以下代码复制并粘贴到其中，这将创建 Amazon Transcribe 客户端对象。将 *REGION* 替换为您的 Amazon 区域。

```
import  { TranscribeClient }  from  "@aws-sdk/client-transcribe";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
```

```
// Create Transcribe service object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

此示例代码可在 GitHub 上的此处找到。

创建文件名为 `transcribe-delete-job.js` 的 Node.js 模块。确保如前所示配置 SDK，包括安装所需的客户端和软件包。使用所需参数创建参数对象，并使用 `DeleteMedicalJobCommand` 命令删除医疗作业。

> **Note**
>
> 将 *JOB_NAME* 替换为要删除的作业的名称。

```
// Import the required AWS SDK clients and commands for Node.js
import { DeleteMedicalTranscriptionJobCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "./libs/transcribeClient.js";

// Set the parameters
export const params = {
  MedicalTranscriptionJobName: "MEDICAL_JOB_NAME", // For example,
 'medical_transciption_demo'
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new DeleteMedicalTranscriptionJobCommand(params)
    );
    console.log("Success - deleted");
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

要运行示例，请在命令提示符中键入以下内容。

```
node transcribe-delete-medical-job.js
```

此示例代码可在 GitHub 上的此处找到。

在 Amazon EC2 实例上设置 Node.js

将 Node.js 与软件开发工具包配合使用的常见场景 JavaScript 是在亚马逊弹性计算云 (Amazon EC2) 实例上设置和运行 Node.js Web 应用程序。在本教程中，您将创建一个 Linux 实例，使用 SSH 连接到该实例，然后安装 Node.js 以在该实例上运行。

先决条件

本教程假定您已经使用公有 DNS 名称启动 Linux 实例，该实例可从 Internet 访问并且您可以使用 SSH 来连接。有关更多信息，请参阅适用于 Linux 实例的 Amazon EC2 用户指南中的第 1 步：启动实例。

Important

在启动新的 Amazon EC2 实例时，请使用 Amazon Linux 2023 Amazon 机器映像 (AMI)。

还必须将安全组配置为允许 SSH（端口 22）、 HTTP（端口 80）和 HTTPS（端口 443）连接。有关这些先决条件的更多信息，请参阅适用于 Linux 实例的 Amazon EC2 用户指南 中的使用 Amazon EC2 进行设置。

过程

以下过程可帮助您在 Amazon Linux 实例上安装 Node.js。您可以使用此服务器来托管 Node.js Web 应用程序。

在 Linux 实例上设置 Node.js

1. 使用 SSH 以 `ec2-user` 身份连接您的 Linux 实例。
2. 通过在命令行中键入以下内容，安装节点版本管理器 (nvm)。

   Warning

   Amazon 不控制以下代码。在运行之前，请务必验证其真实性和完整性。有关此代码的更多信息可以在 nvm GitHub 存储库中找到。

```
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.7/install.sh | bash
```

由于 nvm 可以安装多个版本的 Node.js 并允许您在各个版本之间切换，我们将使用 nvm 安装 Node.js。

3. nvm通过在命令行键入以下内容进行加载。

```
source ~/.bashrc
```

4. 通过在命令行键入以下命令，使用 nvm 安装 Node.js 的最新 LTS 版本。

```
nvm install --lts
```

安装 Node.js 还会安装节点程序包管理器 (npm)，以便您根据需要安装其它模块。

5. 通过在命令行键入以下内容，测试 Node.js 已安装并正确运行。

```
node -e "console.log('Running Node.js ' + process.version)"
```

这将显示以下消息，其中显示正在运行的 Node.js 的版本。

Running Node.js *VERSION*

> **Note**
>
> 节点安装仅适用于当前的 Amazon EC2 会话。如果您重启 CLI 会话，则需要再次使用 nvm 来启用已安装的节点版本。如果实例终止，则需要重新安装节点。另一种方法是在获得要保留的配置后，制作一个 Amazon EC2 实例的 Amazon 机器映像 (AMI)，如以下主题所述。

创建 Amazon 机器映像 (AMI)

在 Amazon EC2 实例上安装 Node.js 后，您可以从该实例创建 Amazon 机器映像 (AMI)。创建 AMI 可通过同一个 Node.js 安装，轻松地预置多个 Amazon EC2 实例。有关更多信息，请参阅适用于 Linux 实例的 Amazon EC2 用户指南中的创建由 Amazon EBS 支持的 Linux AMI。

相关资源

有关本主题中使用的命令和软件的更多信息，请参阅以下网页：

- 节点版本管理器 (nvm)-参见 nvm 存储库。 GitHub
- 节点程序包管理器 (npm)：请参阅 npm 网站。

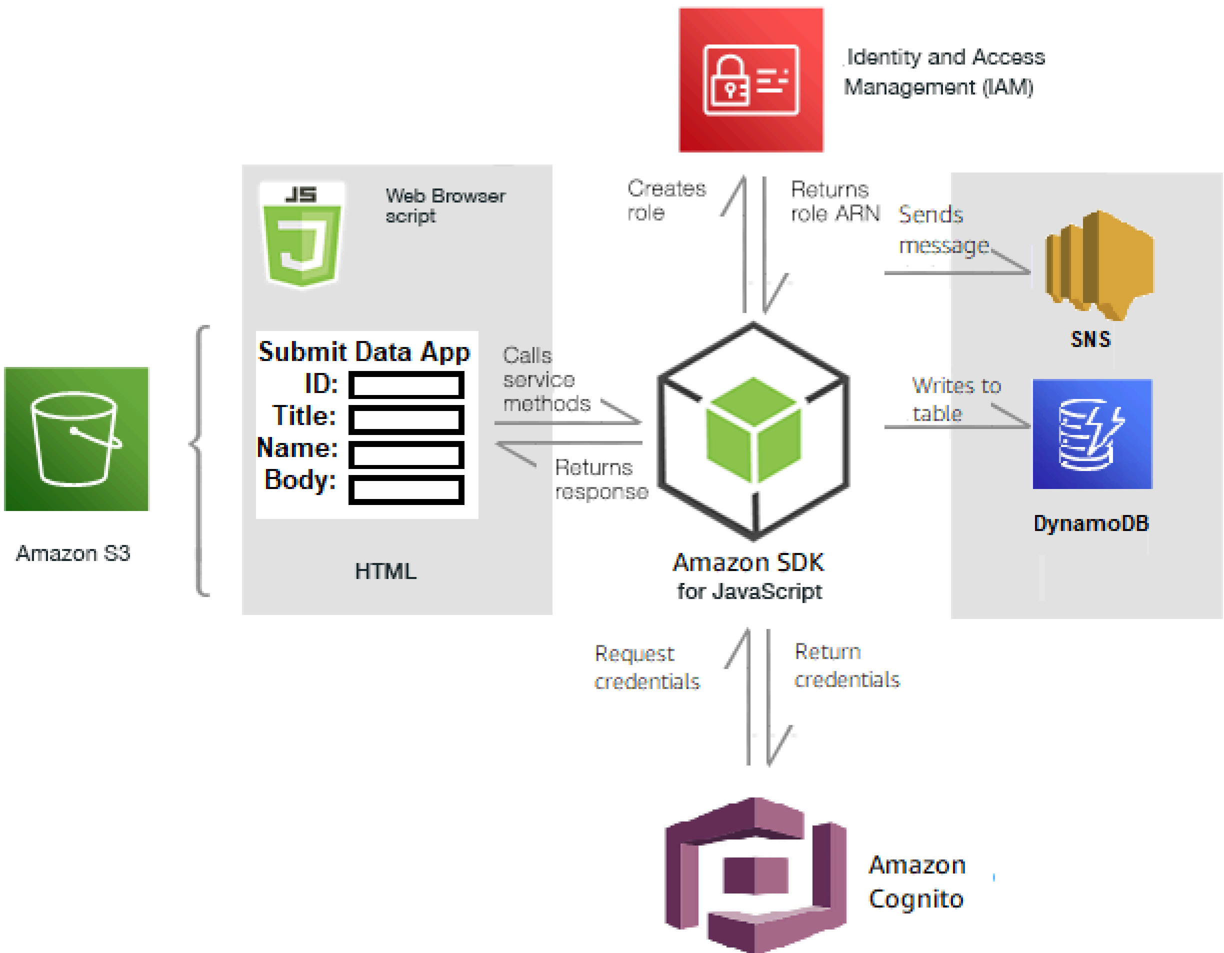
构建应用程序以将数据提交到 DynamoDB

此跨服务 Node.js 教程说明了如何构建一个应用程序，使用户能够向 Amazon DynamoDB 表提交数据。此应用程序使用以下服务：

- Amazon Identity and Access Management (IAM) 和 Amazon Cognito，用于获取授权和权限。
- Amazon DynamoDB (DynamoDB)，用于创建和更新表。
- Amazon Simple Notification Service (Amazon SNS)，用于在用户更新表格时通知应用程序管理员。

情景

在本教程中，一个 HTML 页面提供了一个基于浏览器的应用程序，用于向 Amazon DynamoDB 表提交数据。当用户更新表格时，该应用程序会使用 Amazon SNS 通知应用程序管理员。



构建应用程序：

1. 先决条件
2. 预置资源
3. 创建 HTML
4. 创建浏览器脚本
5. 后续步骤

先决条件

完成以下先决任务：

- 设置项目环境以运行这些 Node TypeScript 示例，并安装所需的 Amazon SDK for JavaScript 和第三方模块。请按照 GitHub 上的说明进行操作。

- 使用用户凭证创建共享配置文件。有关提供共享凭证文件的更多信息，请参阅《Amazon SDK 和工具参考指南》 中的共享配置和凭证文件。

创建 Amazon 资源

此应用程序需要以下资源：

- Amazon Identity and Access Management (IAM)；具有以下权限的未经身份验证的 Amazon Cognito 用户角色：
  - sns:Publish
  - dynamodb:PutItem
- 一个 DynamoDB 表。

您可以在 Amazon 控制台中手动创建这些资源，但我们建议使用本教程中所述的 Amazon CloudFormation 预置这些资源。

使用 Amazon CloudFormation 创建 Amazon

Amazon CloudFormation 让您能够以可预测、可重复的方式创建和预置 Amazon 基础设施部署。有关 Amazon CloudFormation 的更多信息，请参阅 Amazon CloudFormation 用户指南。

使用 Amazon CLI 创建 Amazon CloudFormation 堆栈：

1. 按照 Amazon CLI 用户指南中的说明安装和配置 Amazon CLI。
2. 在项目文件夹的根目录中创建一个名为 `setup.yaml` 的文件，然后将 GitHub 上此处的内容复制到该文件中。

   > **Note**
   >
   > Amazon CloudFormation 模板是使用 GitHub 上此处提供的 Amazon CDK 生成的。有关 Amazon CDK 的更多信息，请参阅 Amazon Cloud Development Kit (Amazon CDK) 开发人员指南。

3. 从命令行运行以下命令，将 *STACK_NAME* 替换为堆栈的唯一名称，将 *REGION* 替换为您的 Amazon 区域。

> **Important**
>
> 在一个 Amazon 区域和一个 Amazon 账户中，堆栈名称必须唯一。您最多可指定 128 个字符，支持数字和连字符。

```
aws cloudformation create-stack --stack-name STACK_NAME --template-body file://
setup.yaml --capabilities CAPABILITY_IAM --region REGION
```

有关 `create-stack` 命令参数的更多信息，请参阅 Amazon CLI 命令参考指南和 Amazon CloudFormation 用户指南。

要查看创建的资源，请在 Amazon 管理控制台中打开 Amazon CloudFormation，选择堆栈，然后选择资源选项卡。

4. 创建堆栈后，使用 Amazon SDK for JavaScript 填充 DynamoDB 表，如填充表中所述。

填充表

要填充表，请先创建一个名为 `libs` 的目录，然后在其中创建一个名为 `dynamoClient.js` 的文件，再将下面的内容粘贴到其中。将 *REGION* 替换为您的 Amazon 区域，然后将 *IDENTITY_POOL_ID* 替换为 Amazon Cognito 身份池 ID。这将创建 DynamoDB 客户端对象。

```
import { CognitoIdentityClient } from "@aws-sdk/client-cognito-identity";
import { fromCognitoIdentityPool } from "@aws-sdk/credential-provider-cognito-
identity";
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

const REGION = "REGION";
const IDENTITY_POOL_ID = "IDENTITY_POOL_ID"; // An Amazon Cognito Identity Pool ID.

// Create an Amazon DynaomDB service client object.
const dynamoClient = new DynamoDBClient({
  region: REGION,
  credentials: fromCognitoIdentityPool({
    client: new CognitoIdentityClient({ region: REGION }),
    identityPoolId: IDENTITY_POOL_ID,
  }),
});
```

```
export { dynamoClient };
```

此代码可以在 GitHub 上的此处找到。

接下来，在项目文件夹中创建一个 dynamoAppHelperFiles 文件夹，在其中创建一个 update-table.js 文件，然后将 GitHub 上此处的内容复制到其中。

```
// Import required AWS SDK clients and commands for Node.js
import { PutItemCommand } from "@aws-sdk/client-dynamodb";
import { dynamoClient } from "../libs/dynamoClient.js";

// Set the parameters
export const params = {
  TableName: "Items",
  Item: {
    id: { N: "1" },
    title: { S: "aTitle" },
    name: { S: "aName" },
    body: { S: "aBody" },
  },
};

export const run = async () => {
  try {
    const data = await dynamoClient.send(new PutItemCommand(params));
    console.log("success");
    console.log(data);
  } catch (err) {
    console.error(err);
  }
};
run();
```

在命令行处，运行以下命令。

```
node update-table.js
```

此代码可以在 GitHub 上的此处找到。

为应用程序创建前端页面

在这里，您可以为应用程序创建前端 HTML 浏览器页面。

创建一个 DynamoDBApp 目录，创建一个名为 index.html 的文件，然后从 GitHub 上的此处将代码复制进去。script 元素可添加 main.js 文件，其中包含该示例所需的所有 JavaScript。您将在本教程的后面部分中创建 main.js 文件。index.html 中的其余代码将创建用于捕获用户输入的数据的浏览器页面。

此示例代码可在 GitHub 上的此处找到。

创建浏览器脚本

首先，创建示例所需的服务客户端对象。创建一个 libs 目录，创建 snsClient.js，并将以下代码粘贴到其中。替换每个对象中的 *REGION* 和 *IDENTITY_POOL_ID*。

> **Note**
>
> 使用您在创建 Amazon 资源 中创建的 Amazon Cognito 身份池的 ID。

```
import { CognitoIdentityClient } from "@aws-sdk/client-cognito-identity";
import { fromCognitoIdentityPool } from "@aws-sdk/credential-provider-cognito-identity";
import { SNSClient } from "@aws-sdk/client-sns";

const REGION = "REGION";
const IDENTITY_POOL_ID = "IDENTITY_POOL_ID"; // An Amazon Cognito Identity Pool ID.

// Create an Amazon Comprehend service client object.
const snsClient = new SNSClient({
  region: REGION,
  credentials: fromCognitoIdentityPool({
    client: new CognitoIdentityClient({ region: REGION }),
    identityPoolId: IDENTITY_POOL_ID,
  }),
});

export { snsClient };
```

此代码可以在 GitHub 上的此处找到。

要为本示例创建浏览器脚本，请在名为 DynamoDBApp 的文件夹中，使用文件名 add_data.js 创建一个 Node.js 模块，然后将以下代码粘贴到其中。submitData 函数会将数据提交到 DynamoDB 表，并使用 Amazon SNS 向应用程序管理员发送短信。

在 submitData 函数中，为目标电话号码、在应用程序界面上输入的值以及 Amazon S3 存储桶的名称声明变量。接下来，创建一个用于向表中添加项目的参数对象。如果所有值都不为空，则 submitData 会将项目添加到表中，然后发送消息。请记得使用 window.submitData = submitData，使该功能可供浏览器使用。

```
// Import required AWS SDK clients and commands for Node.js
import { PutItemCommand } from "@aws-sdk/client-dynamodb";
import { PublishCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";
import { dynamoClient } from "../libs/dynamoClient.js";

export const submitData = async () => {
  //Set the parameters
  // Capture the values entered in each field in the browser (by id).
  const id = document.getElementById("id").value;
  const title = document.getElementById("title").value;
  const name = document.getElementById("name").value;
  const body = document.getElementById("body").value;
  //Set the table name.
  const tableName = "Items";

  //Set the parameters for the table
  const params = {
    TableName: tableName,
    // Define the attributes and values of the item to be added. Adding ' + "" '
 converts a value to
    // a string.
    Item: {
      id: { N: id + "" },
      title: { S: title + "" },
      name: { S: name + "" },
      body: { S: body + "" },
    },
  };
  // Check that all the fields are completed.
  if (id != "" && title != "" && name != "" && body != "") {
    try {
      //Upload the item to the table
      await dynamoClient.send(new PutItemCommand(params));
      alert("Data added to table.");
      try {
        // Create the message parameters object.
        const messageParams = {
```

```
          Message: "A new item with ID value was added to the DynamoDB",
          PhoneNumber: "PHONE_NUMBER", //PHONE_NUMBER, in the E.164 phone number
 structure.
          // For example, ak standard local formatted number, such as (415) 555-2671,
 is +14155552671 in E.164
          // format, where '1' in the country code.
        };
        // Send the SNS message
        const data = await snsClient.send(new PublishCommand(messageParams));
        console.log(
          "Success, message published. MessageID is " + data.MessageId,
        );
      } catch (err) {
        // Display error message if error is not sent
        console.error(err, err.stack);
      }
    } catch (err) {
      // Display error message if item is no added to table
      console.error(
        "An error occurred. Check the console for further information",
        err,
      );
    }
    // Display alert if all field are not completed.
  } else {
    alert("Enter data in each field.");
  }
};
// Expose the function to the browser
window.submitData = submitData;
```

此示例代码可在 GitHub 上的此处找到。

最后，在命令提示符中运行以下命令，将本示例的 JavaScript 捆绑到名为 `main.js` 的文件中：

```
webpack add_data.js --mode development --target web --devtool false -o main.js
```

> **Note**
>
> 有关安装 Webpack 的信息，请参阅将应用程序与 webpack 捆绑在一起。

要运行该应用程序，请在浏览器上打开 `index.html`。

删除资源

如本教程开头所述，请务必在学习本教程时终止您创建的所有资源，以确保系统不会向您收费。您可以通过删除在本教程的创建 Amazon 资源 主题中创建的 Amazon CloudFormation 堆栈来实现此目的，如下所示：

1. 打开 Amazon 管理控制台中的 Amazon CloudFormation。
2. 打开堆栈页面，然后选择堆栈。
3. 选择删除。

有关更多 Amazon 跨服务示例，请参阅 Amazon SDK for JavaScript 跨服务示例。

针对经过身份验证的用户构建转录应用程序

在本教程中，您将学习如何：

- 使用 Amazon Cognito 身份池实施身份验证，以接受与 Amazon Cognito 用户池联合的用户。
- 使用 Amazon Transcribe 在浏览器中转录和显示录音。

情景

该应用程序使用户能够使用唯一的电子邮件和用户名进行注册。确认电子邮件后，他们可以录制语音消息，这些消息会自动转录并显示在应用程序中。

工作方式

该应用程序使用两个 Amazon S3 存储桶，一个用于托管应用程序代码，另一个用于存储转录。该应用程序使用 Amazon Cognito 用户池对您的用户进行身份验证。已经过身份验证的用户有 IAM 权限来访问所需的 Amazon 服务。

用户首次录制语音消息时，Amazon S3 会在 Amazon S3 存储桶中创建一个带有用户名的唯一文件夹，用于存储转录内容。Amazon Transcribe 将语音消息转录为文本，并将其以 JSON 格式保存在用户的文件夹中。当用户刷新应用程序时，其转录内容将会显示，可供下载或删除。

完成本教程大约需要 30 分钟。

步骤

构建应用程序：

1. 先决条件
2. 创建 Amazon 资源
3. 创建 HTML
4. 准备浏览器脚本
5. 运行应用程序
6. 删除资源

先决条件

- 设置项目环境以运行此节点 JavaScript 示例，并安装所需的 Amazon SDK for JavaScript 和第三方模块。请按照 GitHub 上的说明进行操作。
- 使用用户凭证创建共享配置文件。有关提供共享凭证文件的更多信息，请参阅《Amazon SDK 和工具参考指南》中的共享配置和凭证文件。

> **Important**
>
> 此示例使用 ECMAScript6 (ES6)。这需要使用 Node.js 版本 13.x 或更高版本。要下载并安装最新版本的 Node.js，请参阅 Node.js 下载。
> 但是，如果您更喜欢使用 CommonJS 语法，请参阅 JavaScript ES6/CommonJS 语法。

创建 Amazon 资源

本节介绍如何使用 Amazon Cloud Development Kit (Amazon CDK) 为此应用程序预置 Amazon 资源。

> **Note**
>
> Amazon CDK 是一个软件开发框架，使您能够定义云应用程序资源。有关更多信息，请参阅 Amazon Cloud Development Kit (Amazon CDK) 开发人员指南。

要为应用程序创建资源，请使用此处 GitHub 上的模板，使用 Amazon Web Services 管理控制台或 Amazon CLI 创建一个 Amazon CDK 堆栈。有关如何在完成本教程后修改堆栈或删除堆栈及其相关资源的说明，请参阅 GitHub 上的此处。

> **Note**
>
> 在一个 Amazon 区域和一个 Amazon 账户中，堆栈名称必须唯一。您最多可指定 128 个字符，支持数字和连字符。

生成的堆栈将自动预置以下资源。

- 具有经过身份验证的用户角色的 Amazon Cognito 身份池。
- 具有 Amazon S3 和 Amazon Transcribe 权限的 IAM 策略已附加到经过身份验证的用户角色。
- Amazon Cognito 用户池，使用户能够注册和登录应用程序。
- 用于托管应用程序文件的 Amazon S3 存储桶。
- 用于存储转录的 Amazon S3 存储桶。

> **Important**
>
> 此 Amazon S3 存储桶允许读取（列出）公开访问权限，这使任何人都可以列出存储桶中的对象并有可能滥用信息。如果您在完成本教程后没有立即删除此 Amazon S3 存储桶，我们强烈建议您遵守 Amazon Simple Storage Service 用户指南 中的 Amazon S3 安全最佳实践。

创建 HTML

创建一个 `index.html` 文件，然后将以下内容复制并粘贴到文件中。该页面包含用于录制语音消息的按钮面板，以及一个显示当前用户的先前转录消息的表格。body 元素末尾的脚本标签调用 `main.js`，其中包含该应用程序的所有浏览器脚本。您可以使用 Webpack 创建 main.js，如本教程的下一部分所述。

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
```

```
    <title>title</title>
    <link rel="stylesheet" type="text/css" href="recorder.css">
    <style>
        table, td {
            border: 1px solid black;
        }
    </style>
</head>
<body>
<h2>Record</h2>
<p>
    <button id="record" onclick="startRecord()"></button>
    <button id="stopRecord" disabled onclick="stopRecord()">Stop</button>
<p id="demo" style="visibility: hidden;"></p>
</p>
<p>
    <audio id="recordedAudio"></audio>
</p>

<h2>My transcriptions</h2>
<table id="myTable1" style ="width:678px;">
</table>
<table id="myTable" style ="width:678px;">
        <tr>
    <td style = "font-weight:bold">Time created</td>
    <td style = "font-weight:bold">Transcription</td>
    <td style = "font-weight:bold">Download</td>
    <td style = "font-weight:bold">Delete</td>
       </tr>
</table>

<script type="text/javascript" src="./main.js"></script>
</body>

</html>
```

GitHub 的此处提供了此代码示例。

准备浏览器脚本

您需要使用 Webpack 将三个文件 `index.html`、`recorder.js` 和 `helper.js` 捆绑成一个 `main.js`。本节仅详细描述了 `index.js` 中使用 SDK for JavaScript（可在GitHub 上的此处找到）的函数。

Note

您也需要 `recorder.js` 和 `helper.js`，但是由于它们不包含 Node.js 代码，因此分别在 GitHub 上此处和此处的内嵌注释中进行了说明。

首先，定义参数。`COGNITO_ID` 是您在本教程的创建 Amazon 资源 主题中创建的 Amazon Cognito 用户池的端点。其格式为 `cognito-idp.AWS_REGION.amazonaws.com/USER_POOL_ID`。用户池 ID 是 Amazon 凭证令牌中的 *`ID_TOKEN`*，由“helper.js”文件中的 `getToken` 函数从应用程序 URL 中删除。此令牌将传递至 `loginData` 变量，该变量为 Amazon Transcribe 和 Amazon S3 客户端对象提供登录信息。将 *`"REGION"`* 替换为 Amazon 区域，将 *`"BUCKET"`* 替换为您的存储桶，将 *`"IDENTITY_POOL_ID"`* 替换为您为此示例创建的 Amazon Cognito 身份池的示例页面中的 `IdentityPoolId`。这也被传递给每个客户端对象。

```
// Import the required AWS SDK clients and commands for Node.js
import "./helper.js";
import "./recorder.js";
import { CognitoIdentityClient } from "@aws-sdk/client-cognito-identity";
import { fromCognitoIdentityPool } from "@aws-sdk/credential-provider-cognito-
identity";
import {
  CognitoIdentityProviderClient,
  GetUserCommand,
} from "@aws-sdk/client-cognito-identity-provider";
import { S3RequestPresigner } from "@aws-sdk/s3-request-presigner";
import { createRequest } from "@aws-sdk/util-create-request";
import { formatUrl } from "@aws-sdk/util-format-url";
import {
  TranscribeClient,
  StartTranscriptionJobCommand,
} from "@aws-sdk/client-transcribe";
import {
  S3Client,
  PutObjectCommand,
  GetObjectCommand,
  ListObjectsCommand,
  DeleteObjectCommand,
} from "@aws-sdk/client-s3";
import fetch from "node-fetch";

// Set the parameters.
```

```
// 'COGINTO_ID' has the format 'cognito-idp.eu-west-1.amazonaws.com/COGNITO_ID'.
let COGNITO_ID = "COGNITO_ID";
// Get the Amazon Cognito ID token for the user. 'getToken()' is in 'helper.js'.
let idToken = getToken();
let loginData = {
  [COGNITO_ID]: idToken,
};

const params = {
  Bucket: "BUCKET", // The Amazon Simple Storage Solution (S3) bucket to store the
 transcriptions.
  Region: "REGION", // The AWS Region
  identityPoolID: "IDENTITY_POOL_ID", // Amazon Cognito Identity Pool ID.
};

// Create an Amazon Transcribe service client object.
const client = new TranscribeClient({
  region: params.Region,
  credentials: fromCognitoIdentityPool({
    client: new CognitoIdentityClient({ region: params.Region }),
    identityPoolId: params.identityPoolID,
    logins: loginData,
  }),
});

// Create an Amazon S3 client object.
const s3Client = new S3Client({
  region: params.Region,
  credentials: fromCognitoIdentityPool({
    client: new CognitoIdentityClient({ region: params.Region }),
    identityPoolId: params.identityPoolID,
    logins: loginData,
  }),
});
```

HTML 页面加载后，如果这是用户首次登录该应用程序，则 `updateUserInterface` 会在 Amazon S3 存储桶中创建一个带有用户名的文件夹。如果不是首次登录，它会使用用户先前会话中的任何转录更新用户界面。

```
window.onload = async () => {
  // Set the parameters.
  const userParams = {
```

```
    // Get the access token. 'GetAccessToken()' is in 'helper.js'.
    AccessToken: getAccessToken(),
  };
  // Create a CognitoIdentityProviderClient client object.
  const client = new CognitoIdentityProviderClient({ region: params.Region });
  try {
    const data = await client.send(new GetUserCommand(userParams));
    const username = data.Username;
    // Export username for use in 'recorder.js'.
    exports.username = username;
    try {
      // If this is user's first sign-in, create a folder with user's name in Amazon S3
 bucket.
      // Otherwise, no effect.
      const Key = `${username}/`;
      try {
        const data = await s3Client.send(
          new PutObjectCommand({ Key: Key, Bucket: params.Bucket })
        );
        console.log("Folder created for user ", data.Username);
      } catch (err) {
        console.log("Error", err);
      }
      try {
        // Get a list of the objects in the Amazon S3 bucket.
        const data = await s3Client.send(
          new ListObjectsCommand({ Bucket: params.Bucket, Prefix: username })
        );
        // Create a variable for the list of objects in the Amazon S3 bucket.
        const output = data.Contents;
        // Loop through the objects, populating a row on the user interface for each
 object.
        for (var i = 0; i < output.length; i++) {
          var obj = output[i];
          const objectParams = {
            Bucket: params.Bucket,
            Key: obj.Key,
          };
          // Get the name of the object from the Amazon S3 bucket.
          const data = await s3Client.send(new GetObjectCommand(objectParams));
          // Extract the body contents, a readable stream, from the returned data.
          const result = data.Body;
          // Create a variable for the string version of the readable stream.
          let stringResult = "";
```

```
          // Use 'yieldUnit8Chunks' to convert the readable streams into JSON.
          for await (let chunk of yieldUint8Chunks(result)) {
            stringResult += String.fromCharCode.apply(null, chunk);
          }
          // The setTimeout function waits while readable stream is converted into
 JSON.
          setTimeout(function () {
            // Parse JSON into human readable transcript, which will be displayed on
 user interface (UI).
            const outputJSON =
              JSON.parse(stringResult).results.transcripts[0].transcript;
            // Create name for transcript, which will be displayed.
            const outputJSONTime = JSON.parse(stringResult)
              .jobName.split("/")[0]
              .replace("-job", "");
            i++;
            //
            // Display the details for the transcription on the UI.
            // 'displayTranscriptionDetails()' is in 'helper.js'.
            displayTranscriptionDetails(
              i,
              outputJSONTime,
              objectParams.Key,
              outputJSON
            );
          }, 1000);
        }
      } catch (err) {
        console.log("Error", err);
      }
    } catch (err) {
      console.log("Error creating presigned URL", err);
    }
  } catch (err) {
    console.log("Error", err);
  }
};

// Convert readable streams.
async function* yieldUint8Chunks(data) {
  const reader = data.getReader();
  try {
    while (true) {
      const { done, value } = await reader.read();
```

```
      if (done) return;
      yield value;
    }
  } finally {
    reader.releaseLock();
  }
}
```

当用户录制语音消息进行转录时，upload 会将录音上传到 Amazon S3 存储桶。此函数是从 recorder.js 文件中调用的。

```
// Upload recordings to Amazon S3 bucket
window.upload = async function (blob, userName) {
  // Set the parameters for the recording recording.
  const Key = `${userName}/test-object-${Math.ceil(Math.random() * 10 ** 10)}`;
  let signedUrl;

  // Create a presigned URL to upload the transcription to the Amazon S3 bucket when it
 is ready.
  try {
    // Create an Amazon S3RequestPresigner object.
    const signer = new S3RequestPresigner({ ...s3Client.config });
    // Create the request.
    const request = await createRequest(
      s3Client,
      new PutObjectCommand({ Key, Bucket: params.Bucket })
    );
    // Define the duration until expiration of the presigned URL.
    const expiration = new Date(Date.now() + 60 * 60 * 1000);
    // Create and format the presigned URL.
    signedUrl = formatUrl(await signer.presign(request, expiration));
    console.log(`\nPutting "${Key}"`);
  } catch (err) {
    console.log("Error creating presigned URL", err);
  }
  try {
    // Upload the object to the Amazon S3 bucket using a presigned URL.
    response = await fetch(signedUrl, {
      method: "PUT",
      headers: {
        "content-type": "application/octet-stream",
```

```
      },
      body: blob,
    });
    // Create the transcription job name. In this case, it's the current date and time.
    const today = new Date();
    const date =
      today.getFullYear() +
      "-" +
      (today.getMonth() + 1) +
      "-" +
      today.getDate();
    const time =
      today.getHours() + "-" + today.getMinutes() + "-" + today.getSeconds();
    const jobName = date + "-time-" + time;

    // Call the "createTranscriptionJob()" function.
    createTranscriptionJob(
      "s3://" + params.Bucket + "/" + Key,
      jobName,
      params.Bucket,
      Key
    );
  } catch (err) {
    console.log("Error uploading object", err);
  }
};

// Create the AWS Transcribe transcription job.
const createTranscriptionJob = async (recording, jobName, bucket, key) => {
  // Set the parameters for transcriptions job
  const params = {
    TranscriptionJobName: jobName + "-job",
    LanguageCode: "en-US", // For example, 'en-US',
    OutputBucketName: bucket,
    OutputKey: key,
    Media: {
      MediaFileUri: recording, // For example, "https://transcribe-demo.s3-
REGION.amazonaws.com/hello_world.wav"
    },
  };
  try {
    // Start the transcription job.
    const data = await client.send(new StartTranscriptionJobCommand(params));
    console.log("Success - transcription submitted", data);
```

```
  } catch (err) {
    console.log("Error", err);
  }
};
```

deleteTranscription 从用户界面中删除转录，deleteRow 从 Amazon S3 存储桶中删除现有转录。两者均由用户界面上的删除按钮触发。

```
// Delete a transcription from the Amazon S3 bucket.
window.deleteJSON = async (jsonFileName) => {
  try {
    await s3Client.send(
      new DeleteObjectCommand({
        Bucket: params.Bucket,
        Key: jsonFileName,
      })
    );
    console.log("Success - JSON deleted");
  } catch (err) {
    console.log("Error", err);
  }
};
// Delete a row from the user interface.
window.deleteRow = function (rowid) {
  const row = document.getElementById(rowid);
  row.parentNode.removeChild(row);
};
```

最后，在命令提示符中运行以下命令，将本示例的 JavaScript 捆绑到名为 main.js 的文件中：

```
webpack index.js --mode development --target web --devtool false -o main.js
```

Note

有关安装 Webpack 的信息，请参阅将应用程序与 webpack 捆绑在一起。

运行应用程序

您可以在下面的位置查看该应用程序。

```
DOMAIN/login?
client_id=APP_CLIENT_ID&response_type=token&scope=aws.cognito.signin.user.admin+email
+openid+phone+profile&redirect_uri=REDIRECT_URL
```

Amazon Cognito 在 Amazon Web Services 管理控制台中提供链接，支持您轻松运行该应用程序。只需导航到 Amazon Cognito 用户池的应用程序客户端设置，然后选择启动托管 UI 即可。该应用程序的 URL 采用以下格式。

> **Important**
>
> 托管 UI 的响应类型默认为“code”。但是，本教程专为“token”响应类型而设计，因此您必须对其进行更改。

删除 Amazon 资源

完成本教程后，您应删除相关资源，以免产生任何不必要的费用。由于您已将内容添加到两个 Amazon S3 存储桶，因此必须手动将其删除。然后，您可以使用 Amazon Web Services 管理控制台或 Amazon CLI 删除剩余的资源。有关如何修改堆栈或在完成本教程后删除堆栈及其相关资源的说明，请参阅 GitHub 上的此处。

使用 API Gateway 调用 Lambda

您可以通过使用 Amazon API Gateway 来调用 Lambda 函数，Amazon API Gateway 是一项用于大规模创建、发布、维护、监控和保护 REST、HTTP 和 WebSocket API 的 Amazon 服务。API 开发人员可以创建能够访问 Amazon 或其他 Web 服务以及存储在 Amazon 云中的数据的 API。作为 API Gateway 开发人员，您可以创建 API 以在您自己的客户端应用程序中使用。有关更多信息，请参阅什么是 Amazon API Gateway。

Amazon Lambda 是一项计算服务，使您无需预置或管理服务器即可运行代码。您可以使用各种编程语言创建 Lambda 函数。有关 Amazon Lambda 的更多信息，请参阅什么是 Amazon Lambda。

在本示例中，您使用 Lambda JavaScript 运行时 API 创建 Lambda 函数。此示例调用不同的 Amazon 服务以执行特定的应用场景。例如，假设组织在员工入职一周年纪念日时向员工发送移动短信表示祝贺，如下图所示。

Today 2:50 PM

Malcolm happy one year anniversary. We are very happy that you have been working here for a year!

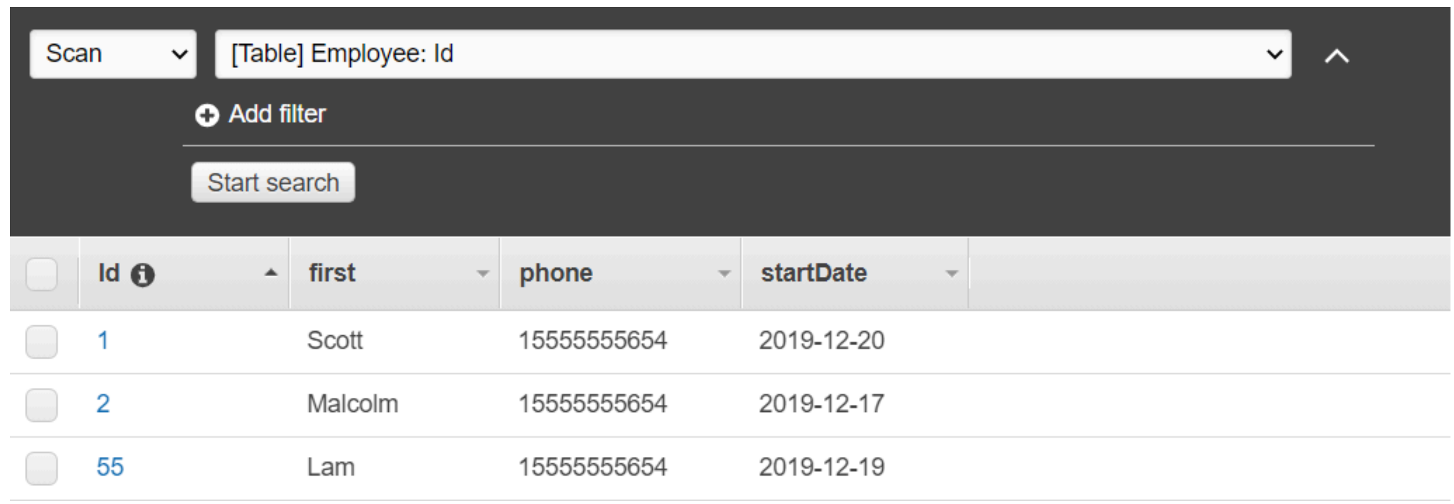
完成此示例大约需要 20 分钟。

此示例向您演示了如何使用 JavaScript 逻辑来创建执行此应用场景的解决方案。例如，您将学习如何使用 Lambda 函数读取数据库以确定哪些员工已达到入职一周年纪念日、如何处理数据以及如何发送短信。然后，您将学习如何使用 API Gateway 通过 Rest 端点调用此 Amazon Lambda 函数。例如，您可以使用以下 curl 命令调用 Lambda 函数：

```
curl -XGET "https://xxxxqjko1o3.execute-api.us-east-1.amazonaws.com/cronstage/
employee"
```

本 Amazon 教程使用名为 Employee 的 Amazon DynamoDB 表，其中包含以下字段。

- id - 表的主键。
- firstName - 员工的名字。
- phone - 员工的电话号码。
- startDate - 员工的入职日期。



Id	first	phone	startDate
1	Scott	15555555654	2019-12-20
2	Malcolm	15555555654	2019-12-17
55	Lam	15555555654	2019-12-19

> **Important**
>
> 完成费用：本文档中包含的 Amazon 服务包含在 Amazon 免费套餐中。但是，请务必在完成此示例后终止所有资源，以确保系统不会向您收费。

构建应用程序：

1. 满足先决条件
2. 创建 Amazon 资源
3. 准备浏览器脚本
4. 创建并上传 Lambda 函数
5. 部署 Lambda 函数
6. 运行应用程序
7. 删除资源

先决条件任务

要设置和运行此示例，您必须先完成以下任务：

- 设置项目环境以运行这些 Node TypeScript 示例，并安装所需的 Amazon SDK for JavaScript 和第三方模块。请按照 GitHub 上的说明进行操作。
- 使用用户凭证创建共享配置文件。有关提供共享凭证文件的更多信息，请参阅《Amazon SDK 和工具参考指南》中的共享配置和凭证文件。

创建 Amazon 资源

本教程要求具有以下资源:

- 一个名为 `Employee` 的 Amazon DynamoDB 表，其键名为 `Id`，其字段如上图所示。请务必输入正确的数据，包括要用来测试此应用场景的有效手机号。有关更多信息，请参阅创建表。
- 具有执行 Lambda 函数的附加权限的 IAM 角色。
- 一个用于托管 Lambda 函数的 Amazon S3 存储桶。

您可以手动创建这些资源，但我们建议使用本教程中所述的 Amazon CloudFormation 预置这些资源。

使用 Amazon CloudFormation 创建 Amazon

Amazon CloudFormation 让您能够以可预测、可重复的方式创建和预置 Amazon 基础设施部署。有关 Amazon CloudFormation 的更多信息，请参阅 Amazon CloudFormation 用户指南。

使用 Amazon CLI 创建 Amazon CloudFormation 堆栈：

1. 按照 Amazon CLI 用户指南中的说明安装和配置 Amazon CLI。
2. 在项目文件夹的根目录中创建一个名为 `setup.yaml` 的文件，然后将 GitHub 上此处的内容复制到该文件中。

   > **Note**
   >
   > Amazon CloudFormation 模板是使用 GitHub 上此处提供的 Amazon CDK 生成的。有关 Amazon CDK 的更多信息，请参阅 Amazon Cloud Development Kit (Amazon CDK) 开发人员指南。

3. 从命令行运行以下命令，将 *STACK_NAME* 替换为堆栈的唯一名称。

   > **Important**
   >
   > 在一个 Amazon 区域和一个 Amazon 账户中，堆栈名称必须唯一。您最多可指定 128 个字符，支持数字和连字符。

   ```
   aws cloudformation create-stack --stack-name STACK_NAME --template-body file://
   setup.yaml --capabilities CAPABILITY_IAM
   ```

   有关 `create-stack` 命令参数的更多信息，请参阅 Amazon CLI 命令参考指南和 Amazon CloudFormation 用户指南。
4. 接下来，按照填充表过程填充表。

填充表

要填充表，请先创建一个名为 `libs` 的目录，然后在其中创建一个名为 `dynamoClient.js` 的文件，再将下面的内容粘贴到其中。

```
const { DynamoDBClient } = require ( "@aws-sdk/client-dynamodb" );
```

```
// Set the AWS Region.
const REGION = "REGION"; // e.g. "us-east-1"
 // Create an Amazon Lambda service client object.
const dynamoClient = new DynamoDBClient({region:REGION});
module.exports = { dynamoClient };
```

此代码可以在 GitHub 上的此处找到。

接下来，在项目文件夹的根目录中创建一个名为 populate-table.js 的文件，然后将 GitHub 上此处的内容复制到该文件中。对于其中一项，将 phone 属性的值替换为 E.164 格式的有效手机号码，将 startDate 的值替换为今天的日期。

在命令行处，运行以下命令。

```
node populate-table.js
```

```

const { BatchWriteItemCommand } = require ( "aws-sdk/client-dynamodb" );
const {dynamoClient} = require ( "./libs/dynamoClient" );

// Set the parameters.
export const params = {
  RequestItems: {
    Employees: [
      {
        PutRequest: {
          Item: {
            id: { N: "1" },
            firstName: { S: "Bob" },
            phone: { N: "155555555555654" },
            startDate: { S: "2019-12-20" },
          },
        },
      },
      {
        PutRequest: {
          Item: {
            id: { N: "2" },
            firstName: { S: "Xing" },
            phone: { N: "155555555555653" },
            startDate: { S: "2019-12-17" },
```

```
          },
        },
      },
      {
        PutRequest: {
          Item: {
            id: { N: "55" },
            firstName: { S: "Harriette" },
            phone: { N: "155555555555652" },
            startDate: { S: "2019-12-19" },
          },
        },
      },
    ],
  },
};

export const run = async () => {
  try {
    const data = await dbclient.send(new BatchWriteItemCommand(params));
    console.log("Success", data);
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

此代码可以在 GitHub 上的此处找到。

创建 Amazon Lambda 函数

配置 SDK

在 libs 目录中，创建名为 snsClient.js 和 lambdaClient.js 的文件，并将以下内容分别粘贴到这些文件中。

```
const  { SNSClient } = require ( "@aws-sdk/client-sns" );
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon SNS service client object.
const snsClient = new SNSClient({ region: REGION });
module.exports = { snsClient };
```

将 *REGION* 替换为 Amazon 区域、此代码可以在 GitHub 上的此处找到。

```
const { LambdaClient } = require ( "@aws-sdk/client-lambda" );
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon Lambda service client object.
const lambdaClient = new LambdaClient({ region: REGION });
module.exports = { lambdaClient };
```

将 *REGION* 替换为 Amazon 区域、此代码可以在 GitHub 上的此处找到。

首先，导入所需的 Amazon SDK for JavaScript (v3) 模块和命令。然后计算今天的日期并将其分配给一个参数。然后，为 `ScanCommand` 创建参数。将 *TABLE_NAME* 替换为您在本示例的 创建 Amazon 资源 部分创建的表的名称。

下面的代码段演示了此步骤。（有关完整示例，请参阅捆绑 Lambda 函数。）

```
"use strict";
const { ScanCommand } = require("@aws-sdk/client-dynamodb");
const { PublishCommand } = require("@aws-sdk/client-sns");
const {snsClient} = require ( "./libs/snsClient" );
const {dynamoClient} = require ( "./libs/dynamoClient" );

// Get today's date.
const today = new Date();
const dd = String(today.getDate()).padStart(2, "0");
const mm = String(today.getMonth() + 1).padStart(2, "0"); //January is 0!
const yyyy = today.getFullYear();
const date = yyyy + "-" + mm + "-" + dd;

// Set the parameters for the ScanCommand method.
const params = {
  // Specify which items in the results are returned.
  FilterExpression: "startDate = :topic",
  // Define the expression attribute value, which are substitutes for the values you
 want to compare.
  ExpressionAttributeValues: {
    ":topic": { S: date },
  },
  // Set the projection expression, which are the attributes that you want.
  ProjectionExpression: "firstName, phone",
  TableName: "Employees",
```

```
};
```

扫描 DynamoDB 表

首先，创建一个名为 sendText 的异步/等待函数，以使用 Amazon SNS PublishCommand 发布短信。然后，添加一个 try 块模式，用于扫描 DynamoDB 表中是否有工作周年纪念日在今天的员工，然后调用 sendText 函数向这些员工发送短信。如果发生错误，则将调用 catch 块。

下面的代码段演示了此步骤。（有关完整示例，请参阅捆绑 Lambda 函数。）

```
// Helper function to send message using Amazon SNS.
exports.handler = async () => {
  // Helper function to send message using Amazon SNS.
  async function sendText(textParams) {
    try {
      await snsClient.send(new PublishCommand(textParams));
      console.log("Message sent");
    } catch (err) {
      console.log("Error, message not sent ", err);
    }
  }
  try {
    // Scan the table to identify employees with work anniversary today.
    const data = await dynamoClient.send(new ScanCommand(params));
    data.Items.forEach(function (element) {
      const textParams = {
        PhoneNumber: element.phone.N,
        Message:
            "Hi " +
            element.firstName.S +
            "; congratulations on your work anniversary!",
      };
      // Send message using Amazon SNS.
      sendText(textParams);
    });
  } catch (err) {
    console.log("Error, could not scan table ", err);
  }
};
```

捆绑 Lambda 函数

本主题介绍如何将此示例的 mylambdafunction.ts 和所需的 Amazon SDK for JavaScript 模块捆绑到名为 index.js 的捆绑文件中。

1. 如果您尚未安装 Webpack，请按照本示例中的先决条件任务部分进行安装。

Note

有关 Webpack 的信息，请参阅将应用程序与 webpack 捆绑在一起。

2. 在命令行中运行以下命令，将本示例的 JavaScript 捆绑到名为 <index.js> 的文件：

```
webpack mylambdafunction.ts --mode development --target node --devtool false --
output-library-target umd -o index.js
```

Important

请注意，输出被命名为 index.js。这是因为 Lambda 函数必须有一个 index.js 处理程序才能工作。

3. 将捆绑的输出文件 index.js 压缩到名为 mylambdafunction.zip 的 ZIP 文件中。
4. 将 mylambdafunction.zip 上传到您在本教程的创建 Amazon 资源 主题中创建的 Amazon S3 存储桶。

部署 Lambda 函数

在项目的根目录中，创建一个 lambda-function-setup.ts 文件，然后将以下内容粘贴到其中。

将 *BUCKET_NAME* 替换为您将 ZIP 版本的 Lambda 函数上传到的 Amazon S3 存储桶的名称。将 *ZIP_FILE_NAME* 替换为您的 ZIP 版本的 Lambda 函数的名称。将 *ROLE* 替换为您在本教程的创建 Amazon 资源 主题中创建的 IAM 角色的 Amazon 资源编号 (ARN)。将 *LAMBDA_FUNCTION_NAME* 替换为 Lambda 函数的名称。

```
// Load the required Lambda client and commands.
const {
  CreateFunctionCommand
} = require ( "@aws-sdk/client-lambda" );
const { lambdaClient} = require ( "./libs/lambdaClient.js );
```

```
// Set the parameters.
const params = {
  Code: {
    S3Bucket: "BUCKET_NAME", // BUCKET_NAME
    S3Key: "ZIP_FILE_NAME", // ZIP_FILE_NAME
  },
  FunctionName: "LAMBDA_FUNCTION_NAME",
  Handler: "index.handler",
  Role: "IAM_ROLE_ARN", // IAM_ROLE_ARN; e.g., arn:aws:iam::650138640062:role/v3-
lambda-tutorial-lambda-role
  Runtime: "nodejs12.x",
  Description:
    "Scans a DynamoDB table of employee details and using Amazon Simple Notification
 Services (Amazon SNS) to " +
    "send employees an email on each anniversary of their start-date.",
};

const run = async () => {
  try {
    const data = await lambdaClient.send(new CreateFunctionCommand(params));
    console.log("Success", data); // successful response
  } catch (err) {
    console.log("Error", err); // an error occurred
  }
};
run();
```

在命令行中输入以下内容以部署 Lambda 函数。

```
node lambda-function-setup.ts
```

GitHub 的此处提供了此代码示例。

配置 API Gateway 以调用 Lambda 函数

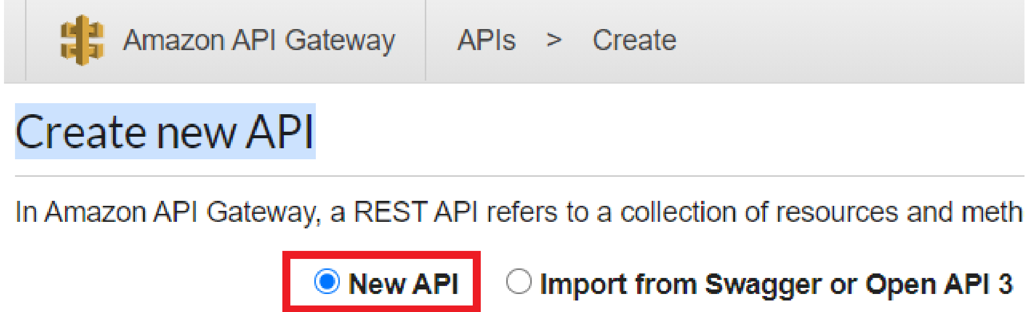
构建应用程序：

1. 创建 rest API
2. 测试 API Gateway 方法
3. 部署 API Gateway 方法

创建 rest API

您可以使用 API Gateway 控制台为 Lambda 函数创建 rest 端点。完成后，您就可以使用 restful 调用来调用 Lambda 函数。

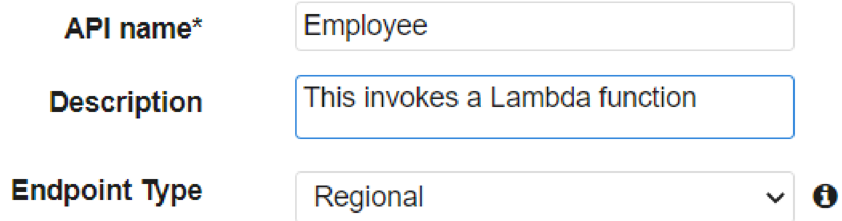
1. 登录 Amazon API Gateway 控制台。
2. 在 Rest API 下，选择生成。
3. 选择新建 API。



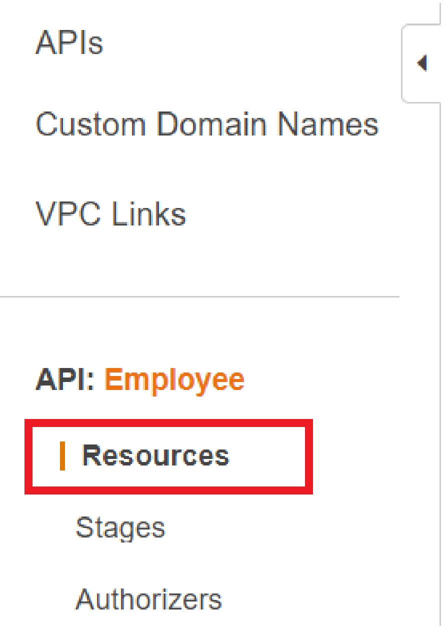
4. 指定 Employee 作为 API 名称并提供描述。

Settings

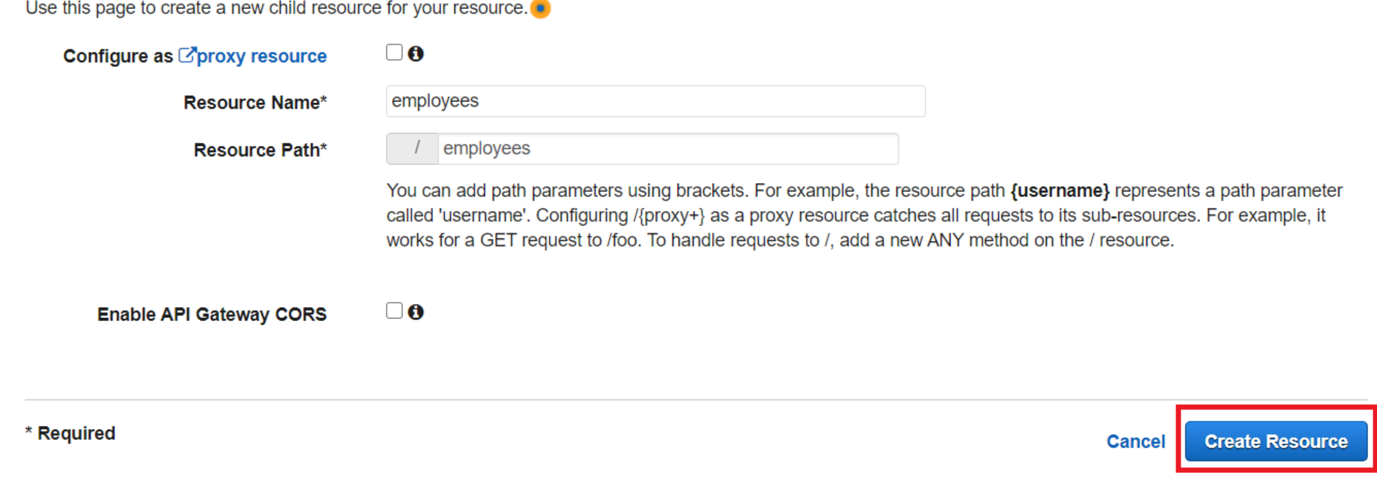
Choose a friendly name and description for your API.



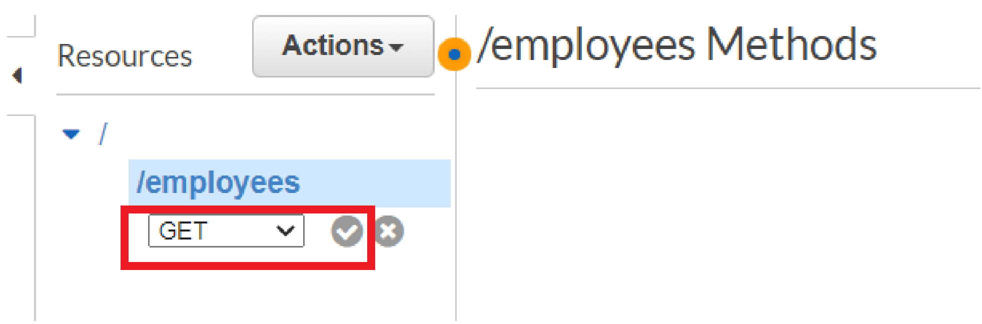
5. 选择创建 API。
6. 在 Employee 部分下选择资源。



7. 在名称字段中，指定员工。
8. 选择创建资源。
9. 从操作下拉菜单中，选择创建资源。



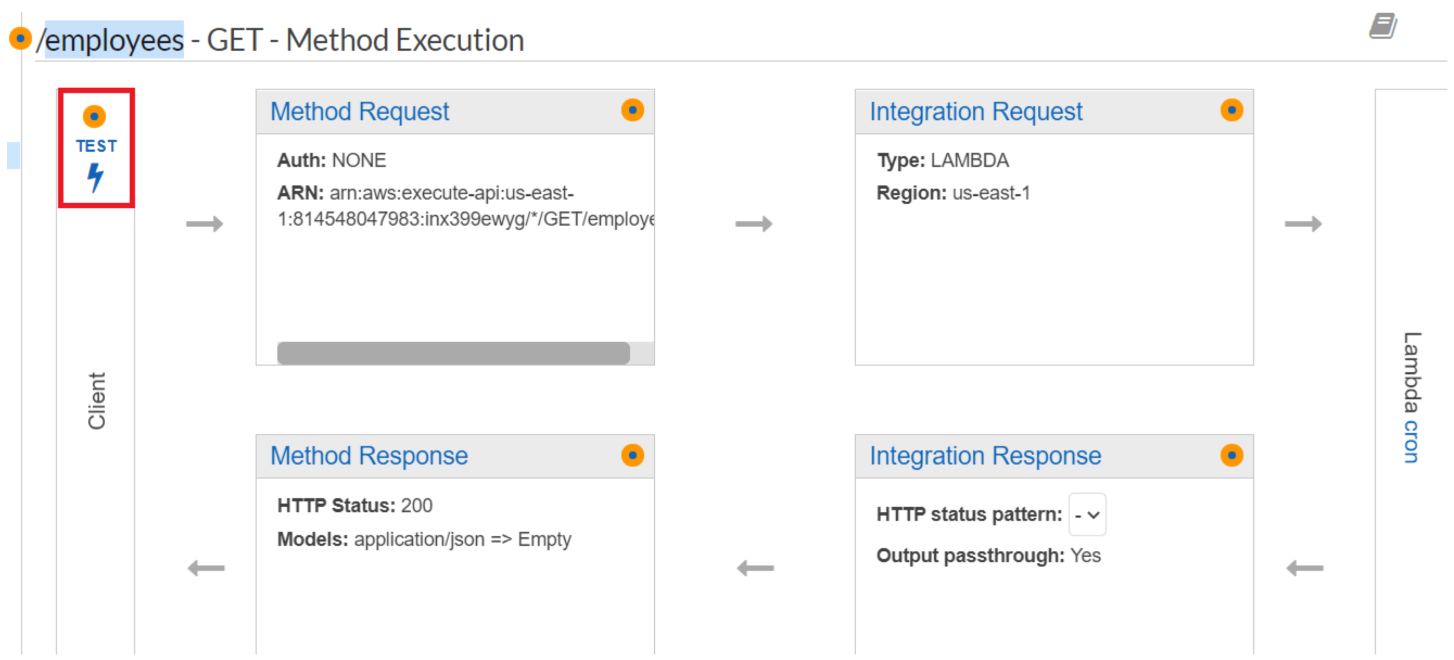
10. 选择 /employees，从操作中选择创建方法，然后从 /employees 下的下拉菜单中选择 GET。选择复选标记图标。



11. 选择 Lambda 函数并输入 mylambdafunction 作为 Lambda 函数名称。选择保存。

测试 API Gateway 方法

在本教程中，您可以测试调用 mylambdafunction Lambda 函数的 API Gateway 方法。要测试该方法，请选择测试，如下图所示。

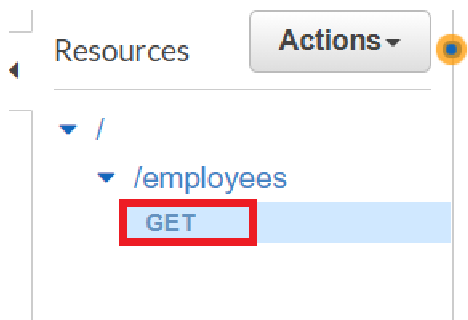


调用 Lambda 函数后，您可以查看日志文件以查看成功消息。

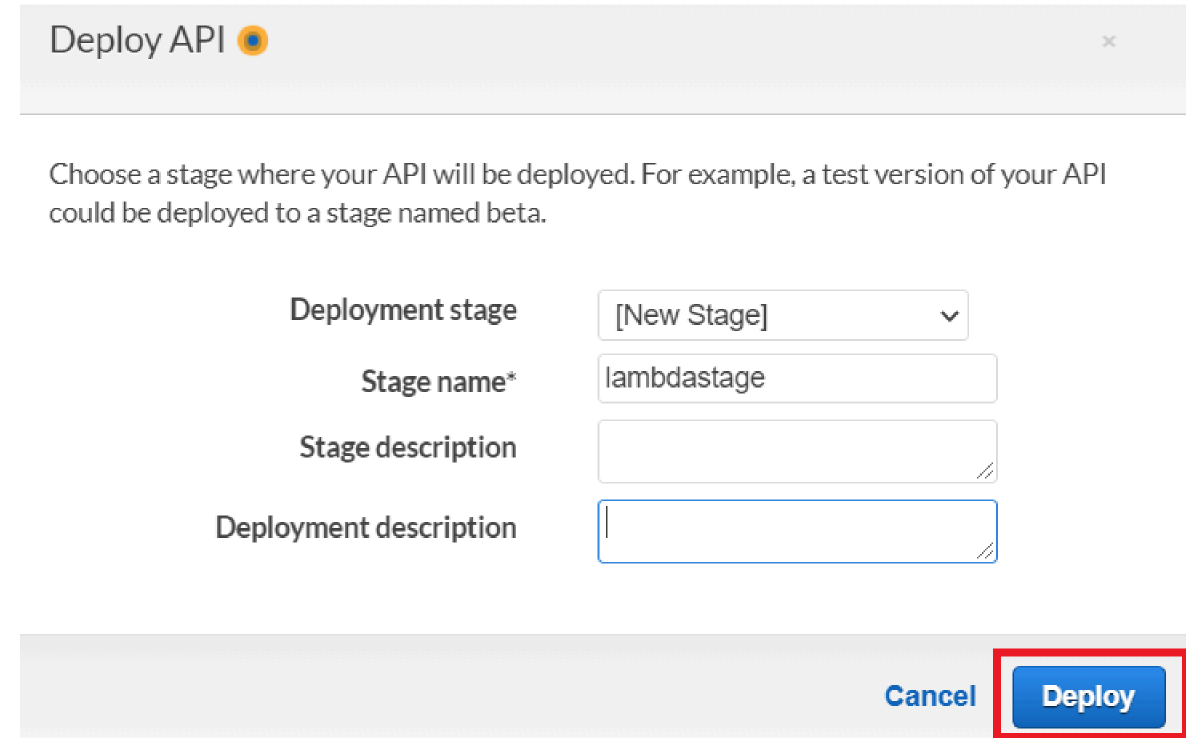
部署 API Gateway 方法

测试成功后，您可以从 Amazon API Gateway 控制台部署该方法。

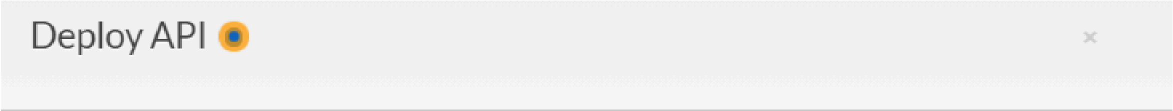
1. 选择 GET。



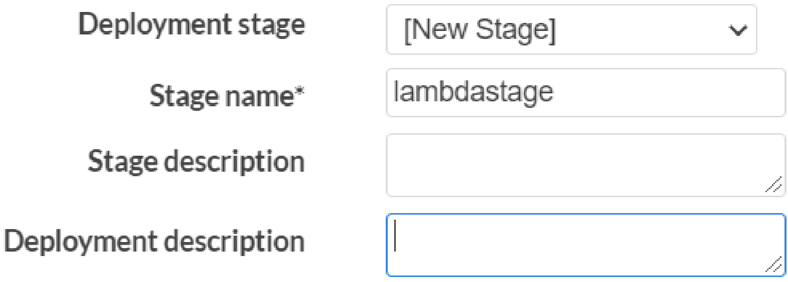
2. 从操作下拉列表中，选择部署 API。

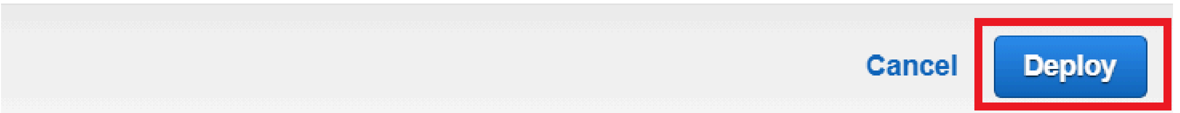


3. 填写部署 API 表单，然后选择部署。

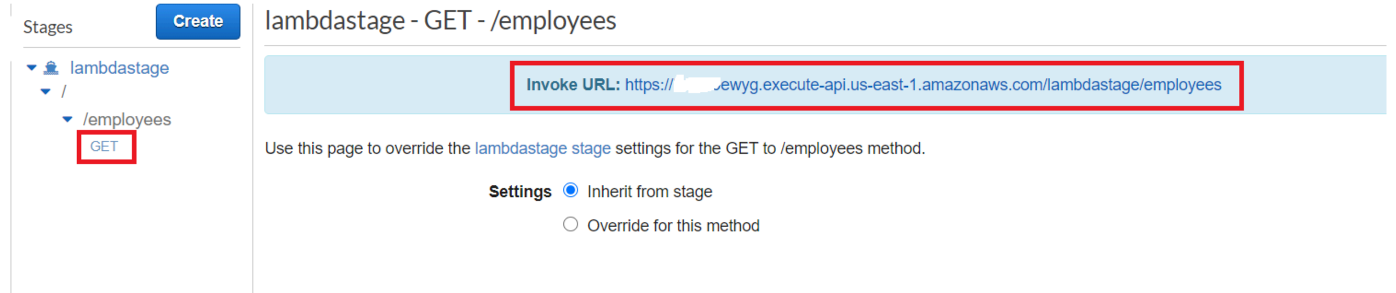


Choose a stage where your API will be deployed. For example, a test version of your API could be deployed to a stage named beta.





4. 选择保存更改。
5. 再次选择 GET，可以看到 URL 已更改。这是您可以用来调用 Lambda 函数的调用 URL。



删除资源

恭喜您！您已使用 Amazon SDK for JavaScript 通过 Amazon API Gateway 调用了 Lambda 函数。如本教程开头所述，请务必在学习本教程时终止您创建的所有资源，以确保系统不会向您收费。您可以通过删除在本教程的创建 Amazon 资源 主题中创建的 Amazon CloudFormation 堆栈来实现此目的，方法如下所示：

1. 打开 Amazon 管理控制台中的 Amazon CloudFormation。
2. 打开堆栈页面，然后选择堆栈。
3. 选择删除。

使用 Amazon SDK for JavaScript 创建 Amazon 无服务器工作流

您可以通过对 Amazon SDK for Java 和 Amazon Step Functions 使用 Step Functions 来创建 Amazon 无服务器工作流。每个工作流步骤都通过使用 Amazon Lambda 函数实现。Lambda 是一项计算服务，使您无需预置或管理服务器即可运行代码。Step Functions 是一项无服务器编排服务，可让您搭配使用 Lambda 函数和其他 Amazon 服务来构建业务关键型应用程序。

> **Note**
>
> 您可以使用各种编程语言创建 Lambda 函数。在本教程中，Lambda 函数是使用 Lambda Java API 实现的。有关 Lambda 的更多信息，请参阅什么是 Lambda。

在本教程中，您将创建一个工作流，为组织创建支持工单。每个工作流步骤都会对工单执行一项操作。本教程介绍如何使用 JavaScript 处理工作流数据。例如，您将学习如何读取传递给工作流的数据，如何在步骤之间传递数据，以及如何从工作流调用 Amazon 服务。

完成费用：本文档中包含的 Amazon 服务包含在 Amazon 免费套餐中。

注意：在学习本教程时，请务必终止您创建的所有资源，以确保系统不再向您收费。

主题

- 先决条件任务
- 创建 Amazon 资源
- 创建工作流
- 创建 Lambda 函数
- 将 Lambda 函数添加到工作流
- 使用 Step Functions 控制台执行工作流
- 删除 Amazon 资源

先决条件任务

要设置和运行此示例，您必须先完成以下任务：

- 设置项目环境以运行这些 Node TypeScript 示例，并安装所需的 Amazon SDK for JavaScript 和第三方模块。请按照 GitHub 上的说明进行操作。

- 使用用户凭证创建共享配置文件。有关提供共享凭证文件的更多信息，请参阅《Amazon SDK 和工具参考指南》中的共享配置和凭证文件。

创建 Amazon 资源

本教程要求具有以下资源。

- 一个名为 Case 的 Amazon DynamoDB 表，其密钥名为 Id。
- 一个名为 `lambda-support` 的 IAM 角色，用于调用 Lambda 函数。该角色的策略使其能够从 Lambda 函数调用 Amazon DynamoDB 和 Amazon Simple Email Service 服务。
- 一个名为 `workflow-support` 的 IAM 角色，用于调用工作流。
- 一个用于托管 Lambda 函数的 Amazon S3 存储桶。

您可以手动创建这些资源，但我们建议使用本教程中所述的 Amazon Cloud Development Kit (Amazon CDK) (Amazon CDK) 预置这些资源。

使用 Amazon CloudFormation 创建 Amazon 资源

Amazon CloudFormation 让您能够以可预测、可重复的方式创建和预置 Amazon 基础设施部署。有关 Amazon CloudFormation 的更多信息，请参阅 Amazon CloudFormation 用户指南。

创建 Amazon CloudFormation 堆栈：

1. 按照 Amazon CLI 用户指南中的说明安装和配置 Amazon CLI。
2. 在项目文件夹的根目录中创建一个名为 `setup.yaml` 的文件，然后将 GitHub 上此处的内容复制到该文件中。

   > Note
   >
   > Amazon CloudFormation 模板是使用 GitHub 上此处提供的 Amazon CDK 生成的。有关 Amazon CDK 的更多信息，请参阅 Amazon Cloud Development Kit (Amazon CDK) 开发人员指南。

3. 从命令行运行以下命令，将 *STACK_NAME* 替换为堆栈的唯一名称。

Important

在一个 Amazon 区域和一个 Amazon 账户中，堆栈名称必须唯一。您最多可指定 128 个字符，支持数字和连字符。

```
aws cloudformation create-stack --stack-name STACK_NAME --template-body file://
setup.yaml --capabilities CAPABILITY_IAM
```

有关 `create-stack` 命令参数的更多信息，请参阅 Amazon CLI 命令参考指南和 Amazon CloudFormation 用户指南。

使用 Amazon Web Services 管理控制台创建 Amazon 资源；

要在控制台中为应用程序创建资源，请按照 Amazon CloudFormation 用户指南中的说明进行操作。使用提供的模板创建一个名为 `setup.yaml` 的文件，然后复制 GitHub 上此处的内容。

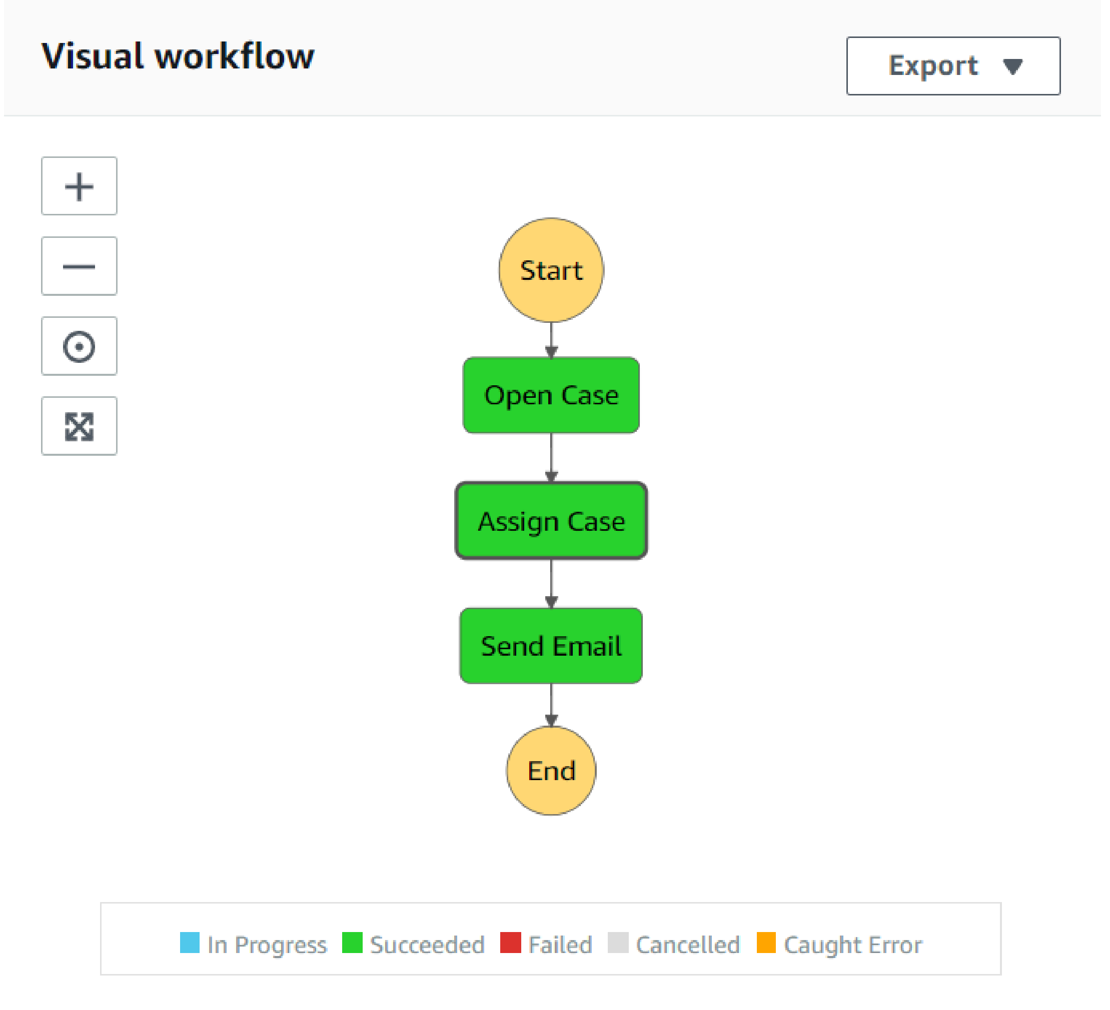
Important

在一个 Amazon 区域和一个 Amazon 账户中，堆栈名称必须唯一。您最多可指定 128 个字符，支持数字和连字符。

在 Amazon CloudFormation 仪表板上打开堆栈，然后选择资源选项卡，即可在控制台中查看资源列表。您将在本教程中需要这些内容。

创建工作流

下图显示了您将使用本教程创建的工作流。



以下是工作流中每个步骤中发生的情况：

+ 开始 - 启动工作流。

+ 开立案例 - 通过将支持工单 ID 值传递给工作流来处理该值。

+ 分配案例 - 将支持案例分配给员工，并将数据存储在 DynamoDB 表中。

+ 发送电子邮件 - 使用 Amazon Simple Email Service (Amazon SES) 向员工发送一封电子邮件，告知他们有一张新工单。

+ 结束 - 停止工作流。

使用 Step Functions 创建无服务器工作流

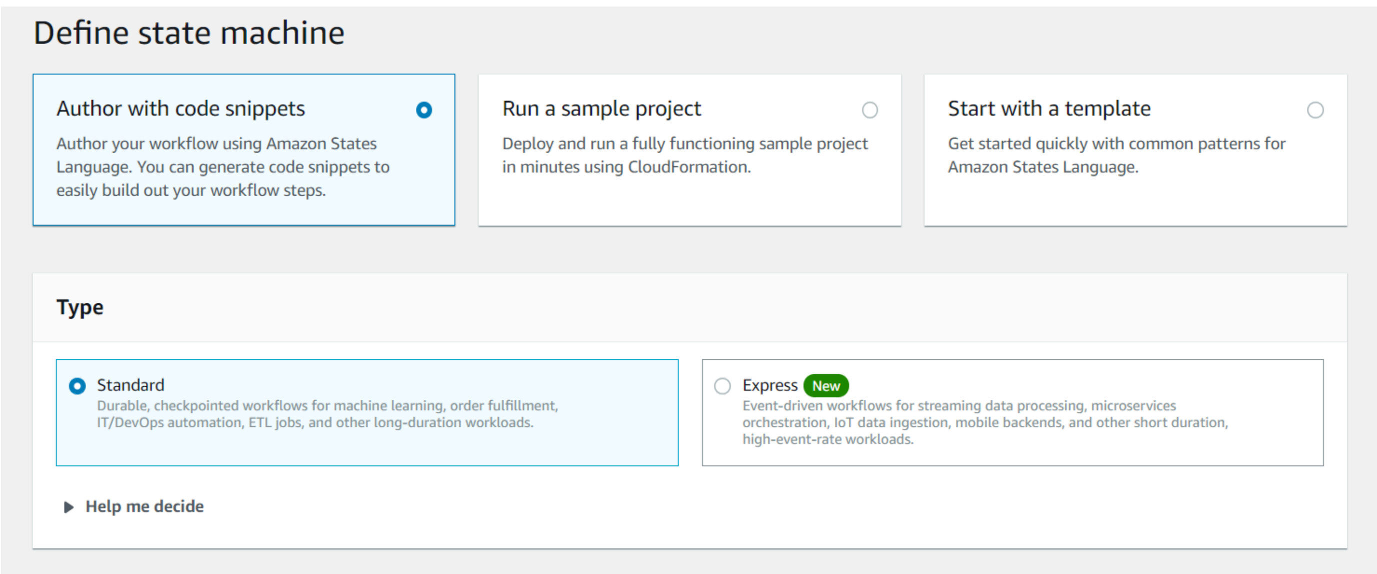
您可以创建处理支持工单的工作流。要使用 Step Functions 定义工作流，您需要创建一个 Amazon States Language（基于 JSON）文档来定义您的状态机。Amazon States Language 文档描述了

每个步骤。定义文档后，Step Functions 将提供工作流的可视化表示。下图显示了 Amazon States Language 文档和工作流的可视化表示。

工作流可以在步骤之间传递数据。例如，开立案例步骤处理案例 ID 值（该值传递给工作流），并将该值传递给分配案例步骤。在本教程的后面部分，您将使用 Lambda 函数创建应用程序逻辑来读取和处理数据值。

创建工作流

1. 打开 [Amazon Web Services 控制台](https://console.aws.amazon.com/)。
2. 选择创建状态机。
3. 选择使用代码段创作。在类型区域中，选择标准。



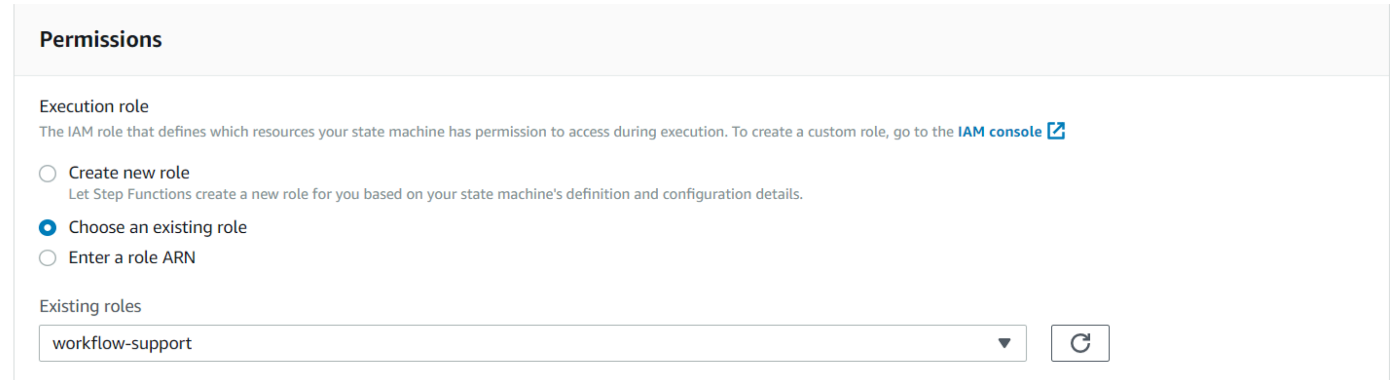
4. 输入以下代码，指定 Amazon States Language 文档。

```
{
"Comment": "A simple Amazon Step Functions state machine that automates a call
 center support session.",
"StartAt": "Open Case",
"States": {
"Open Case": {
"Type": "Task",
"Resource": "arn:aws:lambda:REGION:ACCOUNT_ID:function:FUNCTION_NAME",
"Next": "Assign Case"
},
"Assign Case": {
"Type": "Task",
"Resource": "arn:aws:lambda:REGION:ACCOUNT_ID:function:FUNCTION_NAME",
```

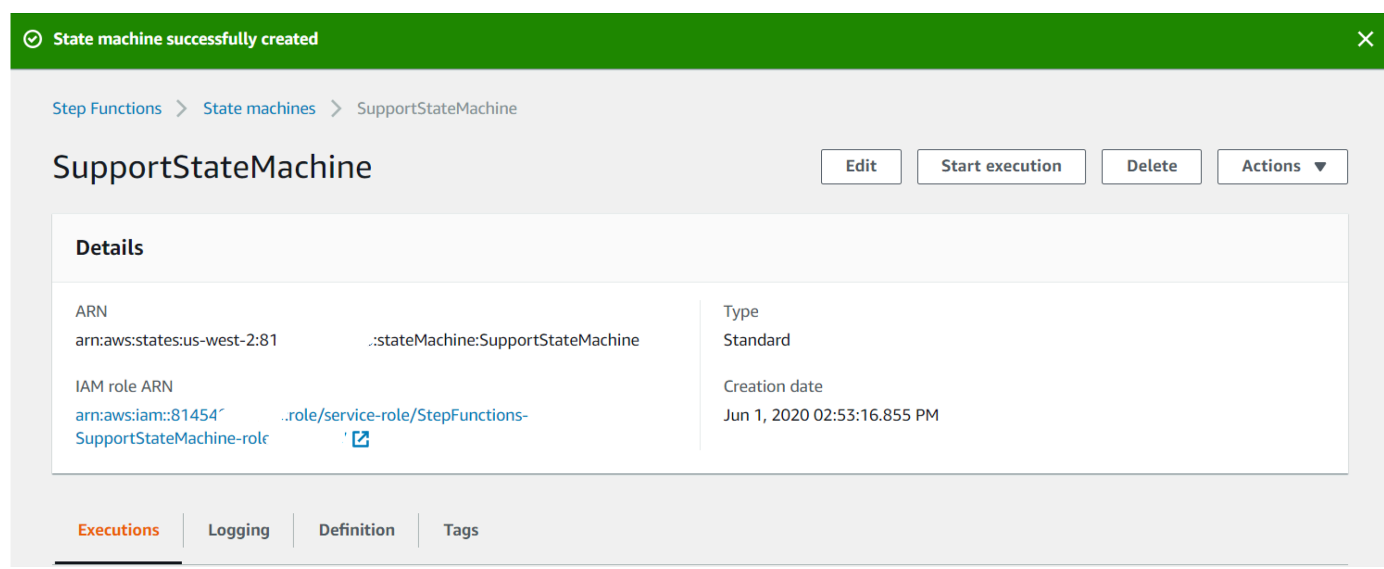
```
"Next": "Send Email"
},
"Send Email": {
"Type": "Task",
"Resource": "arn:aws:lambda:REGION:ACCOUNT_ID:function:FUNCTION_NAME",
"End": true
}
}
}
```

> **Note**
>
> 不用担心与 Lambda 资源值相关的错误。您会在本教程的后面部分中更新这些值。

5. 选择下一步。
6. 在名称字段中，输入 SupportStateMachine。
7. 在权限部分下，选择选择现有角色。
8. 选择 workflow-support（您创建的 IAM 角色）。



9. 选择创建状态机。将出现一条消息，说明状态机已成功创建。



创建 Lambda 函数

使用 Lambda 运行时系统 API 创建 Lambda 函数。在此示例中，有三个工作流步骤，每个步骤对应于每个 Lambda 函数。

创建这些 Lambda 函数，如以下部分所述：

- getId Lambda 函数 - 用作工作流的第一步，用于处理工单 ID 值。
- addItem Lambda 类 - 用作工作流的第二步，将工单分配给员工并将数据存储在 DynamoDB 数据库中。
- sendemail Lambda 类 - 用作工作流的第三步，使用 Amazon SES 向员工发送电子邮件以通知他们有关工单的信息。

getId Lambda 函数

创建一个 Lambda 函数，该函数返回的工单 ID 值将传递至工作流的第二步。

```
exports.handler = async (event) => {
// Create a support case using the input as the case ID, then return a confirmation
 message
try{
    const myCaseID = event.inputCaseID;
    var myMessage = "Case " + myCaseID + ": opened...";
    var result = { Case: myCaseID, Message: myMessage };
```

```
    }
catch(err){
    console.log('Error', err);
    }
};
```

在命令行中输入以下内容，以使用 webpack 将文件捆绑到名为 `index.js` 的文件中。

```
webpack getid.js --mode development --target node --devtool false --output-library-
target umd -o index.js
```

然后将 `index.js` 压缩为一个名为 `getid.js.zip` 的 ZIP 文件。将此 ZIP 文件上传到您在此示例的主题中创建的 Amazon S3 存储桶。

GitHub 的此处提供了此代码示例。

addItem Lambda 类

创建一个 Lambda 函数，用于选择要分配工单的员工，然后将工单数据存储在名为 Case 的 DynamoDB 表中。

```

"use strict";
// Load the required clients and commands.
const { PutItemCommand } = require ( "@aws-sdk/client-dynamodb" );
const { dynamoClient } = require ( "../libs/dynamoClient" );

exports.handler = async (event) => {
  try {
    // Helper function to send message using Amazon SNS.
    const val = event;
    //PersistCase adds an item to a DynamoDB table
    const tmp = Math.random() <= 0.5 ? 1 : 2;
    console.log(tmp);
    if (tmp == 1) {
      const params = {
        TableName: "Case",
        Item: {
          id: { N: val.Case },
          empEmail: { S: "brmur@amazon.com" },
          name: { S: "Tom Blue" },
        },
      };
```

```
      console.log("adding item for tom");
      try {
        const data = await dynamoClient.send(new PutItemCommand(params));
        console.log(data);
      } catch (err) {
        console.error(err);
      }
      var result = { Email: params.Item.empEmail };
      return result;
    } else {
      const params = {
        TableName: "Case",
        Item: {
          id: { N: val.Case },
          empEmail: { S: "RECEIVER_EMAIL_ADDRESS" }, // Valid Amazon Simple
 Notification Services (Amazon SNS) email address.
          name: { S: "Sarah White" },
        },
      };
      console.log("adding item for sarah");
      try {
        const data = await dynamoClient.send(new PutItemCommand(params));
        console.log(data);
      } catch (err) {
        console.error(err);
      }
      return params.Item.empEmail;
      var result = { Email: params.Item.empEmail };
    }
  } catch (err) {
    console.log("Error", err);
  }
};
```

在命令行中输入以下内容，以使用 webpack 将文件捆绑到名为 `index.js` 的文件中。

```
webpack additem.js --mode development --target node --devtool false --output-library-
target umd -o index.js
```

然后将 `index.js` 压缩为一个名为 `additem.js.zip` 的 ZIP 文件。将此 ZIP 文件上传到您在此示例的主题中创建的 Amazon S3 存储桶。

GitHub 的此处提供了此代码示例。

sendemail Lambda 类

创建一个 Lambda 函数，用于向员工发送电子邮件以告知有关新工单的信息。将使用从第二步中传递的电子邮件地址。

```
// Load the required clients and commands.
const { SendEmailCommand } = require ( "@aws-sdk/client-ses" );
const { sesClient } = require ( "../libs/sesClient" );

exports.handler = async (event) => {
  // Enter a sender email address. This address must be verified.
  const senderEmail = "SENDER_EMAIL"
  const sender = "Sender Name <" + senderEmail + ">";

  // AWS Step Functions passes the  employee's email to the event.
  // This address must be verified.
  const recepient = event.S;

  // The subject line for the email.
  const subject = "New case";

  // The email body for recipients with non-HTML email clients.
  const body_text =
    "Hello,\r\n" + "Please check the database for new ticket assigned to you.";

  // The HTML body of the email.
  const body_html = `<html><head></head><body><h1>Hello!</h1><p>Please check the
 database for new ticket assigned to you.</p></body></html>`;

  // The character encoding for the email.
  const charset = "UTF-8";
  var params = {
    Source: sender,
    Destination: {
      ToAddresses: [recepient],
    },
    Message: {
      Subject: {
        Data: subject,
        Charset: charset,
      },
      Body: {
        Text: {
```

```
          Data: body_text,
          Charset: charset,
        },
        Html: {
          Data: body_html,
          Charset: charset,
        },
      },
    },
  };
  try {
    const data = await sesClient.send(new SendEmailCommand(params));
    console.log(data);
  } catch (err) {
    console.error(err);
  }
};
```

在命令行中输入以下内容，以使用 webpack 将文件捆绑到名为 `index.js` 的文件中。

```
webpack sendemail.js --mode development --target node --devtool false --output-library-target umd -o index.js
```

然后将 `index.js` 压缩为一个名为 `sendemail.js.zip` 的 `ZIP` 文件。将此 `ZIP` 文件上传到您在此示例的主题中创建的 Amazon S3 存储桶。

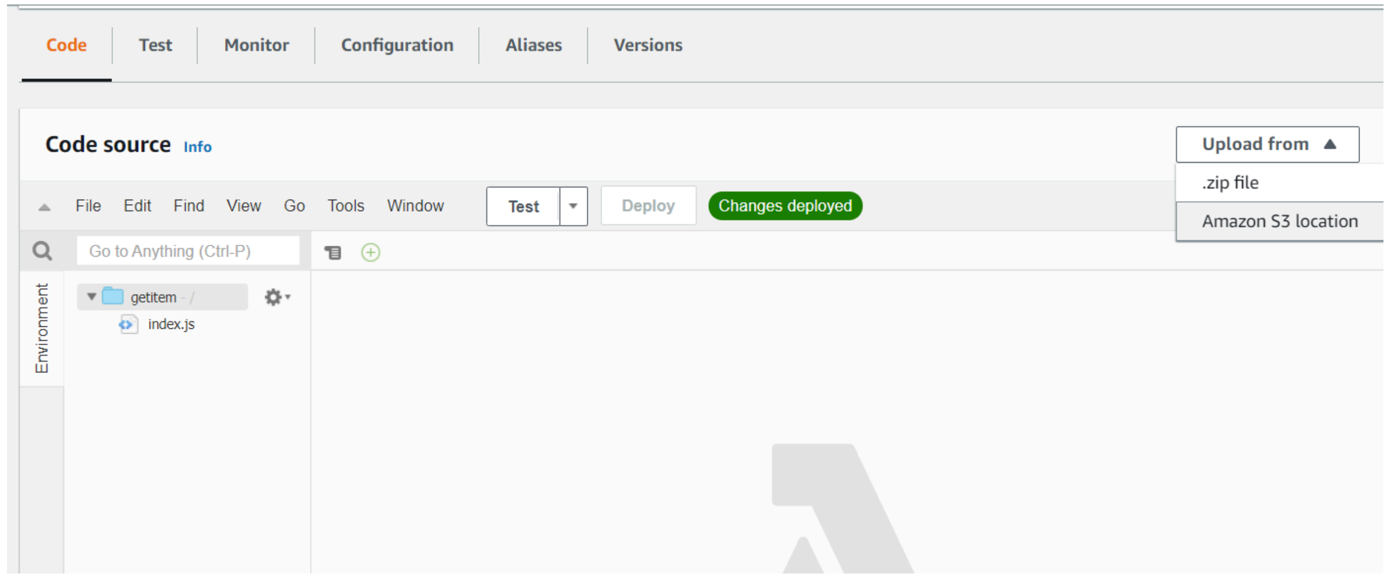
GitHub 的此处提供了此代码示例。

部署 Lambda 函数

部署 `getid` Lambda 函数：

1. 在 Amazon Web Services 控制台中打开 Lambda 控制台。
2. 选择创建函数。
3. 选择从头开始创作。
4. 在基本信息部分中，输入 getid 作为名称。
5. 在运行时系统中，选择 Node.js 14x。
6. 选择使用现有角色，然后选择 lambda-support（您在 中创建的 IAM 角色）。
7. 选择创建函数。

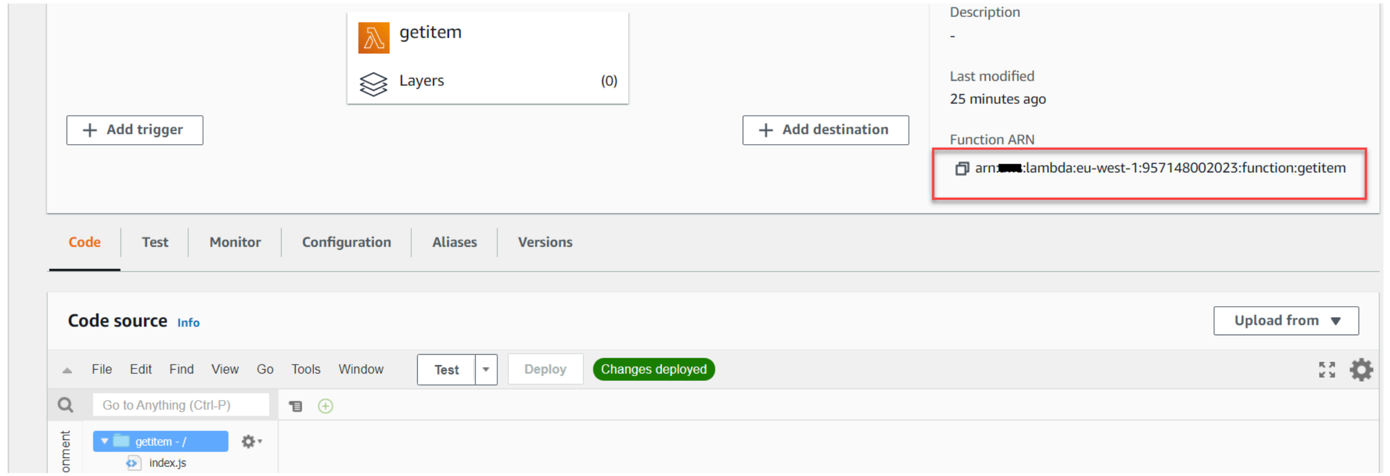
8. 依次选择上传自 - Amazon S3 位置。
9. 选择上传，然后依次选择上传自 - Amazon S3 位置，再输入 Amazon S3 链接 URL。



10. 选择保存。
11. 对新的 Lambda 函数重复执行 additem.js.zip 和 sendemail.js.zip 的过程。完成后，您将拥有三个 Lambda 函数，可以在 Amazon States Language 文档中引用这些函数。

将 Lambda 函数添加到工作流

1. 打开 Lambda 控制台。请注意，您可以在右上角查看 Lambda Amazon 资源名称 (ARN) 值。



2. 复制该值，然后将其粘贴到 Step Functions 控制台中的 Amazon States Language 文档的第 1 步中。
3. 更新分配案例和发送电子邮件步骤的资源。您可以通过这种方式将使用 Amazon SDK for Java 创建的 Lambda 函数连接到使用 Step Functions 创建的工作流中。

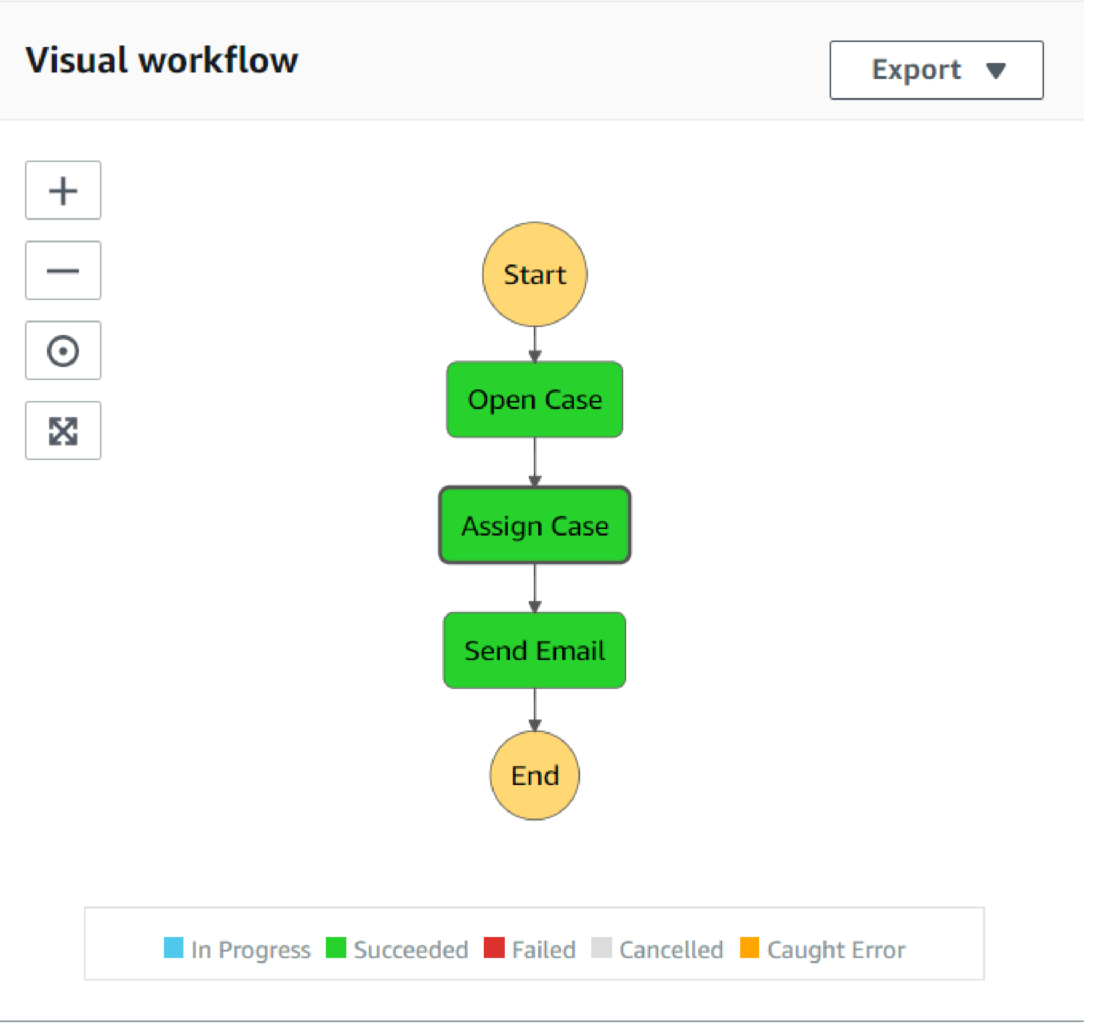
使用 Step Functions 控制台执行工作流

您可以在 Step Functions 控制台上调用工作流。执行会接收 JSON 输入。在此示例中，您可以将以下 JSON 数据传递到工作流。

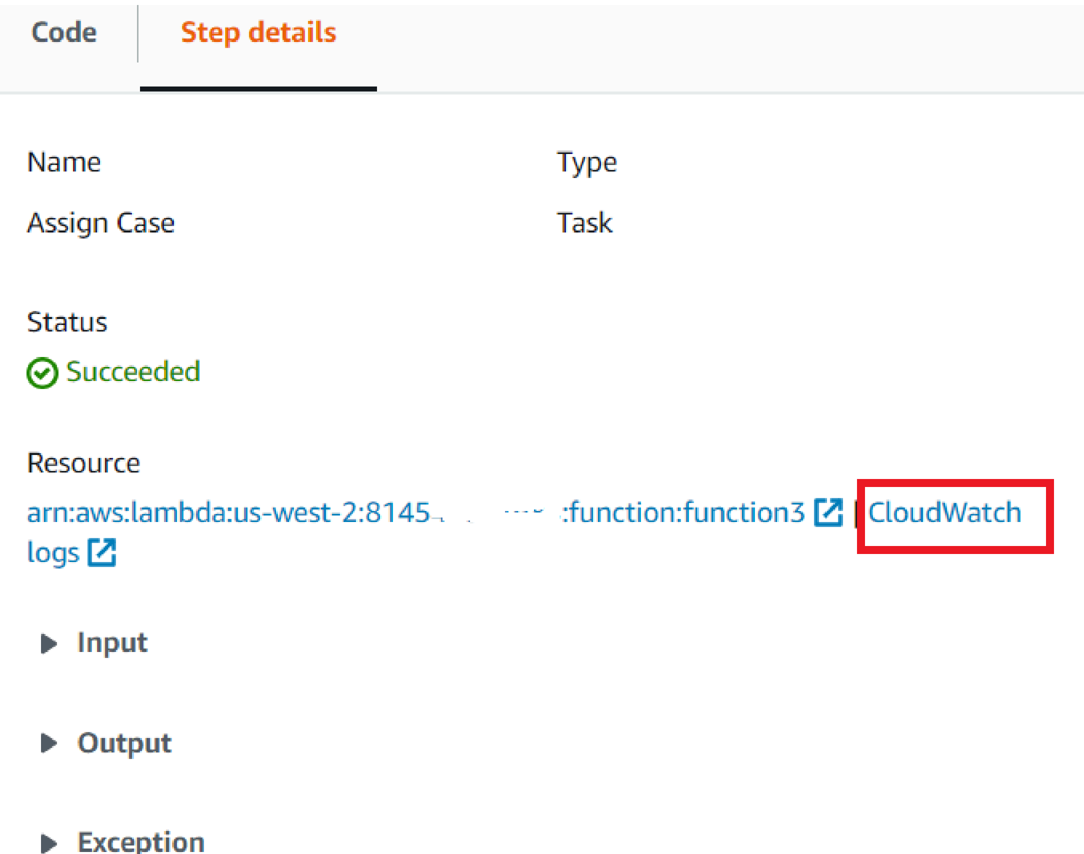
```
{
"inputCaseID": "001"
}
```

执行您的工作流：

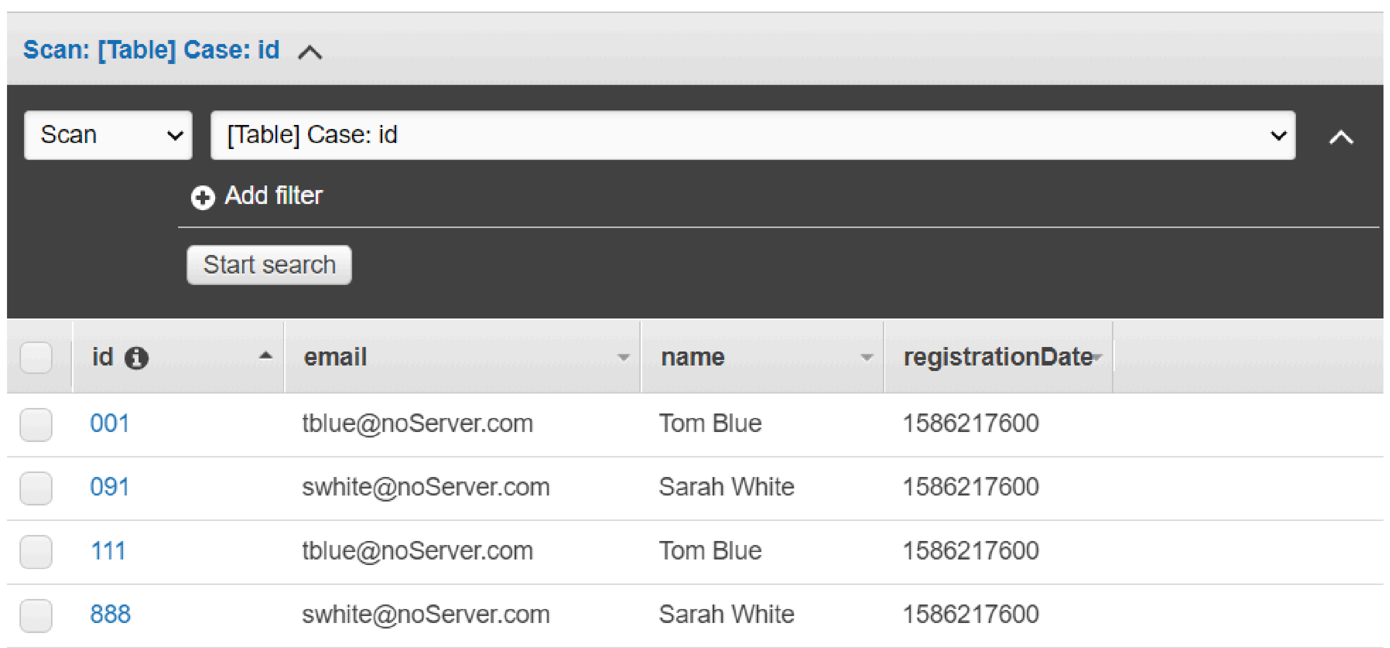
1. 在 Step Functions 控制台上，选择开始执行。
2. 在输入部分中，传递 JSON 数据。查看工作流。每完成一个步骤后，它就会变为绿色。



3. 如果步骤变为红色，则表示出现错误。您可以单击该步骤并查看可从右侧访问的日志。



工作流完成后，您可以查看 DynamoDB 表中的数据。



删除 Amazon 资源

恭喜，您已经使用 Amazon SDK for Java 创建了一个 Amazon 无服务器工作流。如本教程开头所述，请务必在学习本教程时终止您创建的所有资源，以确保系统不会向您收费。您可以通过删除在本教程的创建 Amazon 资源 主题中创建的 Amazon CloudFormation 堆栈来实现此目的，方法如下所示：

1. 打开 Amazon 管理控制台中的 Amazon CloudFormation。
2. 打开堆栈页面，然后选择堆栈。
3. 选择删除。

创建计划事件以执行 Amazon Lambda 函数

您可以使用 Ama CloudWatch zon 事件创建调用Amazon Lambda函数的计划事件。您可以将 CloudWatch 事件配置为使用 cron 表达式来安排何时调用 Lambda 函数。例如，您可以安排一个 CloudWatch 事件，使其在每个工作日调用 Lambda 函数。

Amazon Lambda 是一项计算服务，使您无需预置或管理服务器即可运行代码。您可以使用各种编程语言创建 Lambda 函数。有关 Amazon Lambda 的更多信息，请参阅什么是 Amazon Lambda。

在本教程中，您将使用 Lambda 运行时 API 创建 Lambda 函数。 JavaScript此示例调用不同的 Amazon 服务以执行特定的应用场景。例如，假设组织在员工入职一周年纪念日时向员工发送移动短信表示祝贺，如下图所示。

Today 2:50 PM

Malcolm happy one year anniversary. We are very happy that you have been working here for a year!

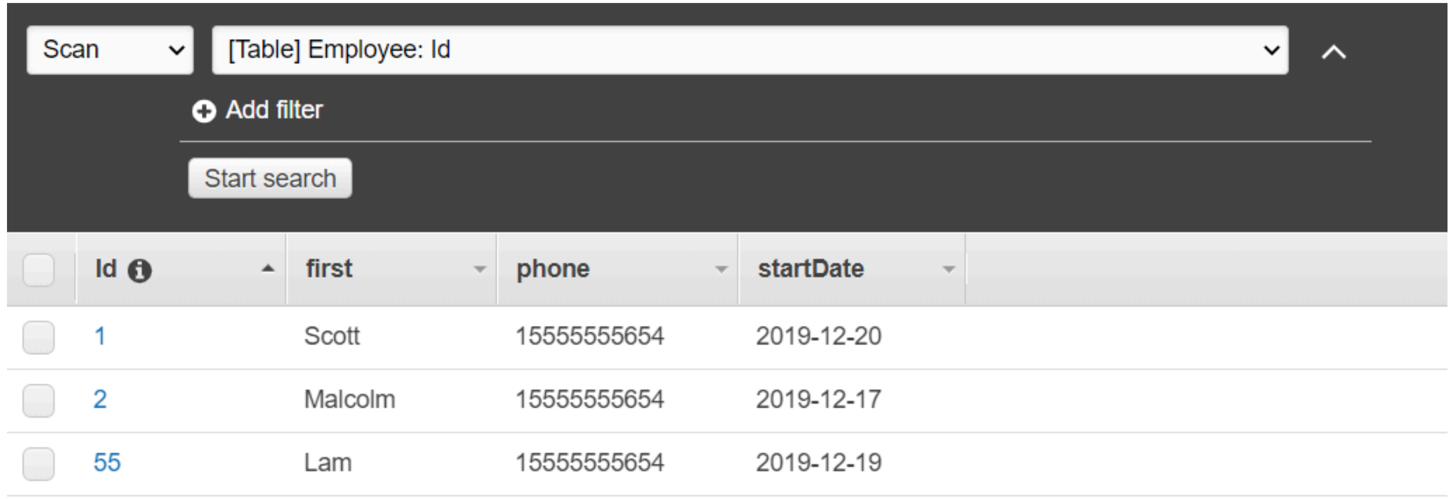
完成本教程大约需要 20 分钟。

本教程向您展示如何使用 JavaScript 逻辑来创建执行此用例的解决方案。例如，您将学习如何使用 Lambda 函数读取数据库以确定哪些员工已达到入职一周年纪念日、如何处理数据以及如何发送短信。然后，您将学习如何使用 cron 表达式在每个工作日调用 Lambda 函数。

本 Amazon 教程使用名为 Employee 的 Amazon DynamoDB 表，其中包含以下字段。

- id - 表的主键。

- firstName - 员工的名字。
- phone - 员工的电话号码。
- startDate - 员工的入职日期。



> **Important**
>
> 完成费用：本文档中包含的 Amazon 服务包含在 Amazon 免费套餐中。但是，请务必在完成本教程后终止所有资源，以确保系统不会向您收费。

构建应用程序：

1. 满足先决条件
2. 创建 Amazon 资源
3. 准备浏览器脚本
4. 创建并上传 Lambda 函数
5. 部署 Lambda 函数
6. 运行应用程序
7. 删除资源

先决条件任务

要设置和运行此示例，您必须先完成以下任务：

- 设置项目环境以运行这些 Node.js TypeScript 示例，并安装所需的模块Amazon SDK for JavaScript 和第三方模块。按照上的说明进行操作 GitHub。
- 使用用户凭证创建共享配置文件。有关提供共享凭证文件的更多信息，请参阅《Amazon SDK 和工具参考指南》中的共享配置和凭证文件。

创建 Amazon 资源

本教程要求具有以下资源。

- 一个名为 Employee 的 Amazon DynamoDB 表，其键名为 Id，其字段如上图所示。请务必输入正确的数据，包括要用来测试此使用案例的有效手机号。有关更多信息，请参阅创建表。
- 具有执行 Lambda 函数的附加权限的 IAM 角色。
- 一个用于托管 Lambda 函数的 Amazon S3 存储桶。

您可以手动创建这些资源，但我们建议使用本教程中所述的 Amazon CloudFormation 预置这些资源。

使用 Amazon CloudFormation 创建 Amazon

Amazon CloudFormation 让您能够以可预测、可重复的方式创建和预置 Amazon 基础设施部署。有关 Amazon CloudFormation 的更多信息，请参阅 Amazon CloudFormation 用户指南。

使用 Amazon CLI 创建 Amazon CloudFormation 堆栈：

1. 按照 Amazon CLI 用户指南中的说明安装和配置 Amazon CLI。
2. 在项目文件夹的根目录`setup.yaml`中创建一个名为的文件，然后将此处的内容复制 GitHub到该文件中。

> **Note**
>
> 该Amazon CloudFormation模板是使用此处Amazon CDK提供的模板生成的 GitHub。有关 Amazon CDK 的更多信息，请参阅 Amazon Cloud Development Kit (Amazon CDK) 开发人员指南。

3. 从命令行运行以下命令，将 *STACK_NAME* 替换为堆栈的唯一名称。

Important

在一个 Amazon 区域和一个 Amazon 账户中，堆栈名称必须唯一。您最多可指定 128 个字符，支持数字和连字符。

```
aws cloudformation create-stack --stack-name STACK_NAME --template-body file://
setup.yaml --capabilities CAPABILITY_IAM
```

有关 `create-stack` 命令参数的更多信息，请参阅 Amazon CLI 命令参考指南和 Amazon CloudFormation 用户指南。

在 Amazon CloudFormation 仪表板上打开堆栈，然后选择资源选项卡，即可在控制台中查看资源列表。您将在本教程中需要这些内容。

4. 创建堆栈后，使用 Amazon SDK for JavaScript 填充 DynamoDB 表，如填充 DynamoDB 表中所述。

填充 DynamoDB 表

要填充表，请先创建一个名为 `libs` 的目录，然后在其中创建一个名为 `dynamoClient.js` 的文件，再将下面的内容粘贴到其中。

```
const { DynamoDBClient } = require( "@aws-sdk/client-dynamodb" );
// Set the AWS Region.
const REGION = "REGION"; // e.g. "us-east-1"
// Create an Amazon DynamoDB service client object.
const dynamoClient = new DynamoDBClient({region:REGION});
module.exports = { dynamoClient };
```

此代码可从此处获得 GitHub。

接下来，在项目文件夹的根目录`populate-table.js`中创建一个名为的文件，然后将此处的内容复制 GitHub到该文件中。对于其中一项，将 phone 属性的值替换为 E.164 格式的有效手机号码，将 `startDate` 的值替换为今天的日期。

在命令行处，运行以下命令。

```
node populate-table.js
```

```
const {
BatchWriteItemCommand } = require( "aws-sdk/client-dynamodb" );
const {dynamoClient} = require(  "./libs/dynamoClient" );
// Set the parameters.
const params = {
  RequestItems: {
    Employees: [
      {
        PutRequest: {
          Item: {
            id: { N: "1" },
            firstName: { S: "Bob" },
            phone: { N: "155555555555654" },
            startDate: { S: "2019-12-20" },
          },
        },
      },
      {
        PutRequest: {
          Item: {
            id: { N: "2" },
            firstName: { S: "Xing" },
            phone: { N: "155555555555653" },
            startDate: { S: "2019-12-17" },
          },
        },
      },
      {
        PutRequest: {
          Item: {
            id: { N: "55" },
            firstName: { S: "Harriette" },
            phone: { N: "155555555555652" },
            startDate: { S: "2019-12-19" },
          },
        },
      },
    ],
  },
};
```

```
export const run = async () => {
  try {
    const data = await dbclient.send(new BatchWriteItemCommand(params));
    console.log("Success", data);
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

此代码可从此处获得 GitHub。

创建 Amazon Lambda 函数

配置 SDK

首先导入所需的 Amazon SDK for JavaScript (v3) 模块和命令：DynamoDBClient 和 DynamoDB ScanCommand，以及 SNSClient 和 Amazon SNS PublishCommand 命令。将 *REGION* 替换为 Amazon 区域、然后计算今天的日期并将其分配给一个参数。然后，为 ScanCommand 创建参数。将 *TABLE_NAME* 替换为您在此示例的创建 Amazon 资源 部分中创建的表的名称。

下面的代码段演示了此步骤。（有关完整示例，请参阅捆绑 Lambda 函数。）

```
"use strict";
// Load the required clients and commands.
const { DynamoDBClient, ScanCommand } = require("@aws-sdk/client-dynamodb");
const { SNSClient, PublishCommand } = require("@aws-sdk/client-sns");

//Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"

// Get today's date.
const today = new Date();
const dd = String(today.getDate()).padStart(2, "0");
const mm = String(today.getMonth() + 1).padStart(2, "0"); //January is 0!
const yyyy = today.getFullYear();
const date = yyyy + "-" + mm + "-" + dd;

// Set the parameters for the ScanCommand method.
const params = {
  // Specify which items in the results are returned.
  FilterExpression: "startDate = :topic",
```

```
  // Define the expression attribute value, which are substitutes for the values you
 want to compare.
  ExpressionAttributeValues: {
    ":topic": { S: date },
  },
  // Set the projection expression, which the the attributes that you want.
  ProjectionExpression: "firstName, phone",
  TableName: "TABLE_NAME",
};
```

扫描 DynamoDB 表

首先，创建一个名为 sendText 的异步/等待函数，以使用 Amazon SNS PublishCommand 发布短信。然后，添加一个 try 块模式，用于扫描 DynamoDB 表中是否有工作周年纪念日在今天的员工，然后调用 sendText 函数向这些员工发送短信。如果发生错误，则将调用 catch 块。

下面的代码段演示了此步骤。（有关完整示例，请参阅捆绑 Lambda 函数。）

```
exports.handler = async (event, context, callback) => {
  // Helper function to send message using Amazon SNS.
  async function sendText(textParams) {
    try {
      const data = await snsclient.send(new PublishCommand(textParams));
      console.log("Message sent");
    } catch (err) {
      console.log("Error, message not sent ", err);
    }
  }
  try {
    // Scan the table to check identify employees with work anniversary today.
    const data = await dbclient.send(new ScanCommand(params));
    data.Items.forEach(function (element, index, array) {
      const textParams = {
        PhoneNumber: element.phone.N,
        Message:
          "Hi " +
          element.firstName.S +
          "; congratulations on your work anniversary!",
      };
      // Send message using Amazon SNS.
      sendText(textParams);
    });
  } catch (err) {
```

```
    console.log("Error, could not scan table ", err);
  }
};
```

捆绑 Lambda 函数

本主题介绍如何将此示例的 `mylambdafunction.js` 和所需的 Amazon SDK for JavaScript 模块捆绑到名为 `index.js` 的捆绑文件中。

1. 如果您尚未安装 Webpack，请按照本示例中的先决条件任务部分进行安装。

> Note
>
> 有关 webpack 的信息，请参阅将应用程序与 webpack 捆绑在一起。

2. 在命令行中运行以下命令，将本示例的捆绑到名 JavaScript 为的文件中`<index.js>`：

```
webpack mylamdbafunction.js --mode development --target node --devtool false --
output-library-target umd -o index.js
```

> Important
>
> 请注意，输出被命名为 `index.js`。这是因为 Lambda 函数必须有一个 `index.js` 处理程序才能工作。

3. 将捆绑的输出文件 `index.js` 压缩到名为 `my-lambda-function.zip` 的 ZIP 文件中。
4. 将 `mylambdafunction.zip` 上传到您在本教程的创建 Amazon 资源 主题中创建的 Amazon S3 存储桶。

以下是 `mylambdafunction.js` 的完整的浏览器脚本代码。

```
"use strict";
// Load the required clients and commands.
const { DynamoDBClient, ScanCommand } = require("@aws-sdk/client-dynamodb");
const { SNSClient, PublishCommand } = require("@aws-sdk/client-sns");

//Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"

// Get today's date.
```

```
const today = new Date();
const dd = String(today.getDate()).padStart(2, "0");
const mm = String(today.getMonth() + 1).padStart(2, "0"); //January is 0!
const yyyy = today.getFullYear();
const date = yyyy + "-" + mm + "-" + dd;

// Set the parameters for the ScanCommand method.
const params = {
  // Specify which items in the results are returned.
  FilterExpression: "startDate = :topic",
  // Define the expression attribute value, which are substitutes for the values you
 want to compare.
  ExpressionAttributeValues: {
    ":topic": { S: date },
  },
  // Set the projection expression, which the the attributes that you want.
  ProjectionExpression: "firstName, phone",
  TableName: "TABLE_NAME",
};

// Create the client service objects.
const dbclient = new DynamoDBClient({ region: REGION });
const snsclient = new SNSClient({ region: REGION });

exports.handler = async (event, context, callback) => {
  // Helper function to send message using Amazon SNS.
  async function sendText(textParams) {
    try {
      const data = await snsclient.send(new PublishCommand(textParams));
      console.log("Message sent");
    } catch (err) {
      console.log("Error, message not sent ", err);
    }
  }
  try {
    // Scan the table to check identify employees with work anniversary today.
    const data = await dbclient.send(new ScanCommand(params));
    data.Items.forEach(function (element, index, array) {
      const textParams = {
        PhoneNumber: element.phone.N,
        Message:
          "Hi " +
          element.firstName.S +
          "; congratulations on your work anniversary!",
```

```
      };
      // Send message using Amazon SNS.
      sendText(textParams);
    });
  } catch (err) {
    console.log("Error, could not scan table ", err);
  }
};
```

部署 Lambda 函数

在项目的根目录中，创建一个 `lambda-function-setup.js` 文件，然后将以下内容粘贴到其中。

将 *BUCKET_NAME* 替换为您将 ZIP 版本的 Lambda 函数上传到的 Amazon S3 存储桶的名称。将 *ZIP_FILE_NAME* 替换为您的 ZIP 版本的 Lambda 函数的名称。将 *IAM_ROLE_ARN* 替换为您在本教程的创建 Amazon 资源 主题中创建的 IAM 角色的 Amazon 资源编号 (ARN)。将 *LAMBDA_FUNCTION_NAME* 替换为 Lambda 函数的名称。

```
// Load the required Lambda client and commands.
const {
   CreateFunctionCommand,
} = require("@aws-sdk/client-lambda");
const {
   lambdaClient
} = require("..libs/lambdaClient.js");

// Instantiate an Lambda client service object.
const lambda = new LambdaClient({ region: REGION });

// Set the parameters.
const params = {
  Code: {
    S3Bucket: "BUCKET_NAME", // BUCKET_NAME
    S3Key: "ZIP_FILE_NAME", // ZIP_FILE_NAME
  },
  FunctionName: "LAMBDA_FUNCTION_NAME",
  Handler: "index.handler",
  Role: "IAM_ROLE_ARN", // IAM_ROLE_ARN; e.g., arn:aws:iam::650138640062:role/v3-
lambda-tutorial-lambda-role
  Runtime: "nodejs12.x",
  Description:
    "Scans a DynamoDB table of employee details and using Amazon Simple Notification
 Services (Amazon SNS) to " +
```

```
    "send employees an email the each anniversary of their start-date.",
};

const run = async () => {
  try {
    const data = await lambda.send(new CreateFunctionCommand(params));
    console.log("Success", data); // successful response
  } catch (err) {
    console.log("Error", err); // an error occurred
  }
};
run();
```

在命令行中输入以下内容以部署 Lambda 函数。

```
node lambda-function-setup.js
```

此代码示例可在此处找到 GitHub。

配置 CloudWatch 为调用 Lambda 函数

CloudWatch 要配置为调用 Lambda 函数，请执行以下操作：

1. 打开 Lambda 控制台的函数页面。
2. 选择 Lambda 函数。
3. 在设计器下方，选择添加触发器。
4. 将触发器类型设置为 CloudWatch Events/ EventBridge。
5. 对于“规则”，选择创建新规则。
6. 填写“规则名称”和“规则描述”。
7. 对于规则类型，请选择计划表达式。
8. 在计划表达式字段中，输入一个 cron 表达式。例如，cron(0 12 ? * MON-FRI *)。
9. 选择添加。

> Note
>
> 有关更多信息，请参阅将 Lambda 与事件配合 CloudWatch 使用。

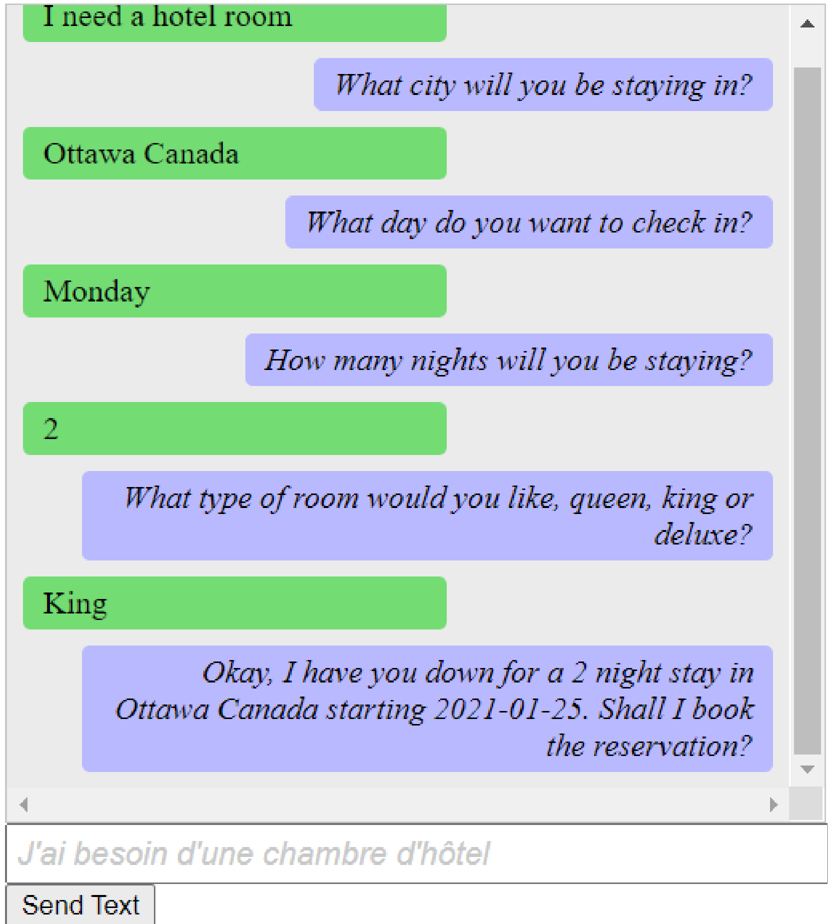
删除资源

恭喜您！您已使用亚马逊 CloudWatch 计划的事件调用了 Lambda 函数。Amazon SDK for JavaScript 如本教程开头所述，请务必在学习本教程时终止您创建的所有资源，以确保系统不会向您收费。您可以通过删除在本教程的创建 Amazon 资源 主题中创建的 Amazon CloudFormation 堆栈来实现此目的，方法如下所示：

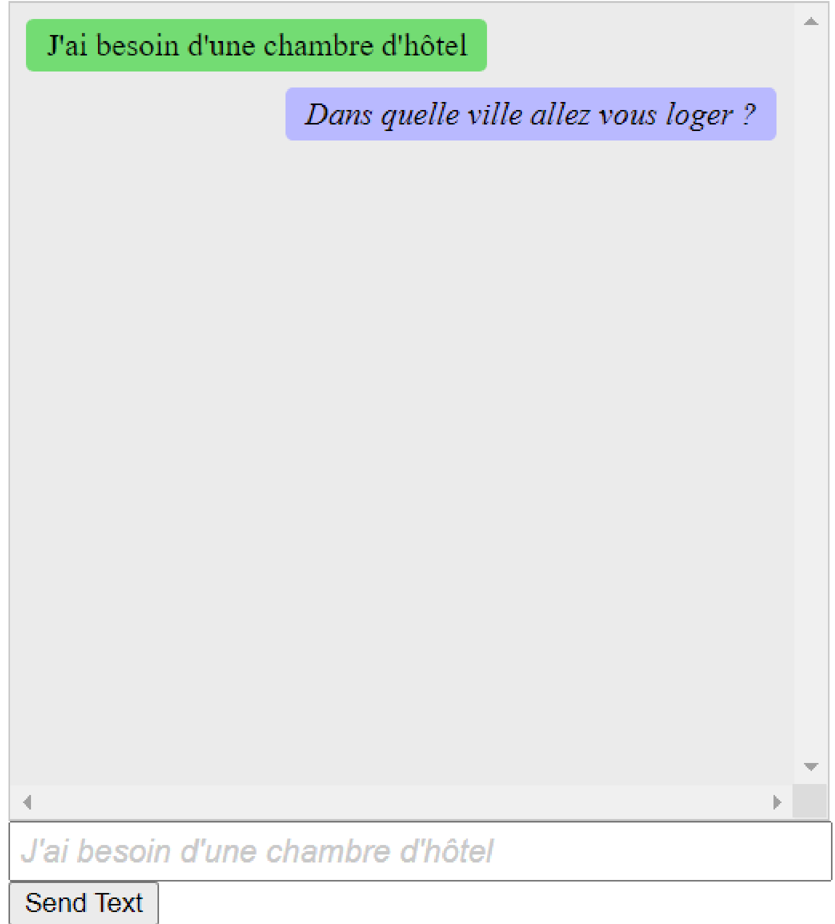
1. 打开Amazon CloudFormation控制台。
2. 在堆栈页面上，选择堆栈。
3. 选择 删除。

构建 Amazon Lex 聊天机器人

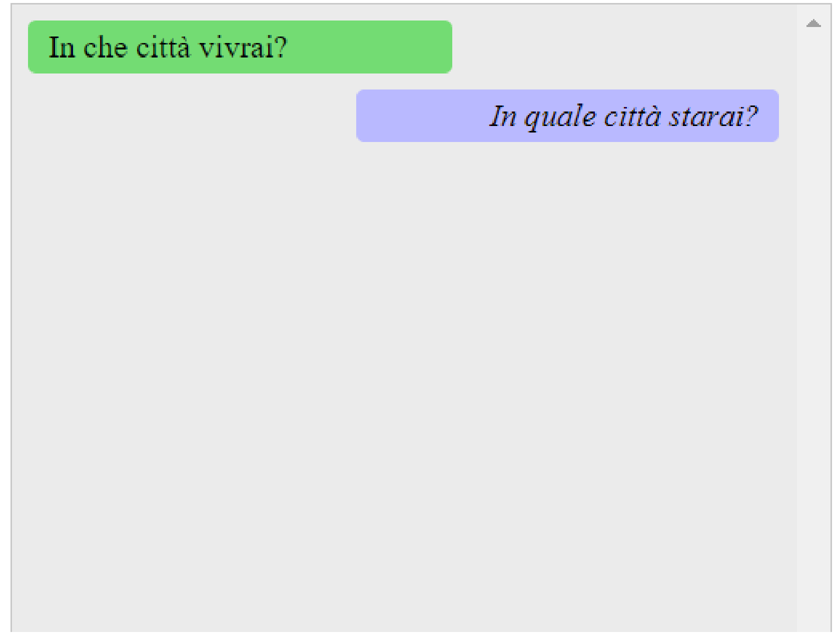
您可以在 Web 应用程序中创建 Amazon Lex 聊天机器人来吸引您的网站访问者。Amazon Lex 聊天机器人是一种无需与人直接接触即可与用户进行在线聊天对话的功能。例如，下图显示了一个 Amazon Lex 聊天机器人，该聊天机器人针对预订酒店房间与用户进行交流。



在本 Amazon 教程中创建的 Amazon Lex 聊天机器人能够处理多种语言。例如，说法语的用户可以输入法语文本并收到以法语返回的回复。



同样，用户可以用意大利语与 Amazon Lex 聊天机器人进行交流。



本 Amazon 教程将指导您创建 Amazon Lex 聊天机器人并将其集成到 Node.js Web 应用程序中。Amazon SDK for JavaScript(v3) 用于调用以下Amazon服务：

- Amazon Lex
- Amazon Comprehend
- Amazon Translate

完成费用：本文档中包含的 Amazon 服务包含在 Amazon 免费套餐中。

注意：在学习本教程时，请务必终止您创建的所有资源，以确保系统不会向您收费。

构建应用程序：

1. 先决条件
2. 预置资源
3. 创建 Amazon Lex 聊天机器人
4. 创建 HTML

5. 创建浏览器脚本
6. 后续步骤

先决条件

要设置和运行此示例，您必须先完成以下任务：

- 设置项目环境以运行这些 Node TypeScript 示例，并安装所需的模块Amazon SDK for JavaScript和第三方模块。按照上的说明进行操作 GitHub。
- 使用用户凭证创建共享配置文件。有关提供共享凭证文件的更多信息，请参阅《Amazon SDK 和工具参考指南》中的共享配置和凭证文件。

> **Important**
>
> 此示例使用 ECMAScript6 (ES6)。这需要使用 Node.js 版本 13.x 或更高版本。要下载并安装最新版本的 Node.js，请参阅 Node.js 下载。
> 但是，如果您更喜欢使用 CommonJS 语法，请参阅 JavaScript ES6/CommonJS 语法。

创建 Amazon 资源

本教程要求具有以下资源。

- 一个未经身份验证的 IAM 角色，附加了以下权限：
  - Amazon Comprehend
  - Amazon Translate
  - Amazon Lex

您可以手动创建这些资源，但我们建议使用本教程中所述的 Amazon CloudFormation 预置这些资源。

使用 Amazon CloudFormation 创建 Amazon

Amazon CloudFormation 让您能够以可预测、可重复的方式创建和预置 Amazon 基础设施部署。有关 Amazon CloudFormation 的更多信息，请参阅 Amazon CloudFormation 用户指南。

使用 Amazon CLI 创建 Amazon CloudFormation 堆栈：

1. 按照 Amazon CLI 用户指南中的说明安装和配置 Amazon CLI。
2. 在项目文件夹的根目录`setup.yaml`中创建一个名为的文件，然后将此处的内容复制 GitHub到该文件中。

> **Note**
>
> 该Amazon CloudFormation模板是使用此处Amazon CDK提供的模板生成的 GitHub。有关 Amazon CDK 的更多信息，请参阅 Amazon Cloud Development Kit (Amazon CDK) 开发人员指南。

3. 从命令行运行以下命令，将 *STACK_NAME* 替换为堆栈的唯一名称。

> **Important**
>
> 在一个 Amazon 区域和一个 Amazon 账户中，堆栈名称必须唯一。您最多可指定 128 个字符，支持数字和连字符。

```
aws cloudformation create-stack --stack-name STACK_NAME --template-body file://
setup.yaml --capabilities CAPABILITY_IAM
```

有关 `create-stack` 命令参数的更多信息，请参阅 Amazon CLI 命令参考指南和 Amazon CloudFormation 用户指南。

要查看创建的资源，请打开 Amazon Lex 控制台，选择堆栈，然后选择资源选项卡。

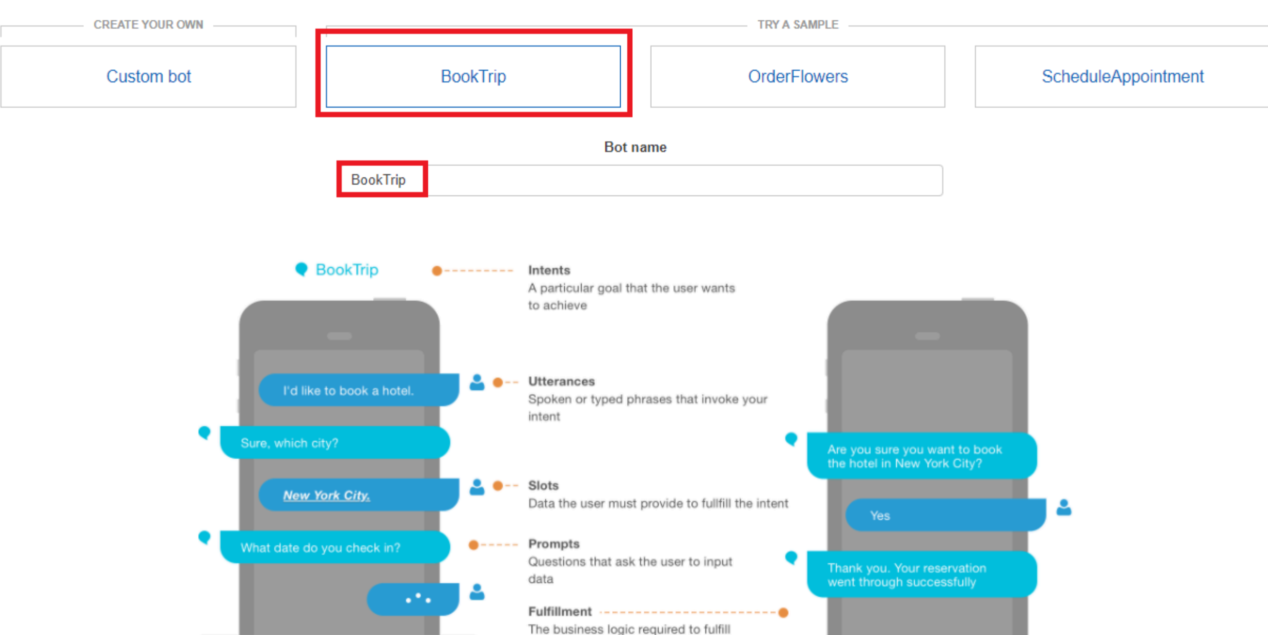
创建 Amazon Lex 机器人

> **Important**
>
> 使用 Amazon Lex 控制台的 V1 创建机器人。此示例不适用于使用 V2 创建的机器人。

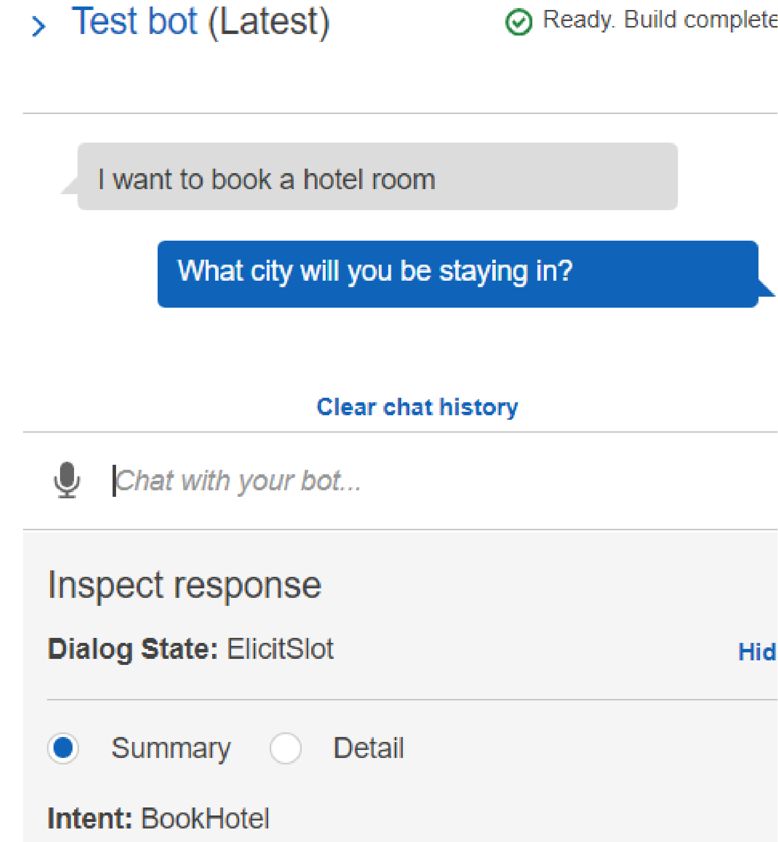
第一步是使用 Amazon Web Services 管理控制台创建 Amazon Lex 聊天机器人。在本示例中，使用了 Amazon Lex BookTrip示例。有关更多信息，请参阅 BookTrip。

- 在 Amazon Web Services 控制台上登录 Amazon Web Services 管理控制台并打开 Amazon Lex 控制台。

- 在“机器人”页面上，选择创建。
- 选择BookTrip蓝图（保留默认的机器人名称 BookTrip）。



- 填写默认设置并选择创建（控制台显示BookTrip机器人）。在“编辑器”选项卡上，查看预配置目的的详细信息。
- 在测试窗口中测试机器人。输入我想预订酒店房间，，开始测试。



- 选择发布并指定别名（使用 Amazon SDK for JavaScript 时您将需要此值）。

> **Note**
>
> 你需要在 JavaScript 代码中引用机器人名称和机器人别名。

创建 HTML

创建一个名为 `index.html`的文件。将以下代码复制并粘贴到 `index.html`。此 HTML 引用 `main.js`。这是 index.js 的捆绑版本，其中包含所需的 Amazon SDK for JavaScript 模块。您将在 [创建 HTML](#) 中创建此文件。 `index.html` 也引用 `style.css`，后者用于添加样式。

```
<!doctype html>
<head>
  <title>Amazon Lex - Sample Application (BookTrip)</title>
  <link type="text/css" rel="stylesheet" href="style.css" />
</head>
```

```
<body>
  <h1 id="title">Amazon Lex - BookTrip</h1>
  <p id="intro">
    This multiple language chatbot shows you how easy it is to incorporate
    <a
      href="https://aws.amazon.com/lex/"
      title="Amazon Lex (product)"
      target="_new"
      >Amazon Lex</a
    >
    into your web apps. Try it out.
  </p>
  <div id="conversation"></div>
  <input
    type="text"
    id="wisdom"
    size="80"
    value=""
    placeholder="J'ai besoin d'une chambre d'hôtel"
  />
  <br />
  <button onclick="createResponse()">Send Text</button>
  <script type="text/javascript" src="./main.js"></script>
</body>
```

此代码也可以在此处找到 GitHub。

创建浏览器脚本

创建一个名为 `index.js`的文件。将以下代码复制并粘贴到 `index.js`。导入所需的 Amazon SDK for JavaScript 模块和命令。为 Amazon Lex、Amazon Comprehend 和 Amazon Translate 创建客户端。将 *REGION* 替换为 Amazon 区域，将 *IDENTITY_POOL_ID* 替换为您在创建 Amazon 资源 中创建的身份池的 ID。要检索此身份池 ID，请在 Amazon Cognito 控制台中打开身份池，选择编辑身份池，然后在侧面菜单中选择示例代码。身份池 ID 将在控制台中以红色文本显示。

首先，创建一个 `libs` 目录，然后通过创建三个文件 `comprehendClient.js`、`lexClient.js` 和 `translateClient.js` 来创建所需的服务客户端对象。将下面的相应代码粘贴到每个文件中，然后在每个文件中替换 *REGION* 和 *IDENTITY_POOL_ID*。

Note

使用您在使用 Amazon CloudFormation 创建 Amazon中创建的 Amazon Cognito 身份池的 ID。

```
import { CognitoIdentityClient } from "@aws-sdk/client-cognito-identity";
import { fromCognitoIdentityPool } from "@aws-sdk/credential-provider-cognito-
identity";
import { ComprehendClient } from "@aws-sdk/client-comprehend";

const REGION = "REGION";
const IDENTITY_POOL_ID = "IDENTITY_POOL_ID"; // An Amazon Cognito Identity Pool ID.

// Create an Amazon Comprehend service client object.
const comprehendClient = new ComprehendClient({
  region: REGION,
  credentials: fromCognitoIdentityPool({
    client: new CognitoIdentityClient({ region: REGION }),
    identityPoolId: IDENTITY_POOL_ID,
  }),
});

export { comprehendClient };
```

```
import { CognitoIdentityClient } from "@aws-sdk/client-cognito-identity";
import { fromCognitoIdentityPool } from "@aws-sdk/credential-provider-cognito-
identity";
import { LexRuntimeServiceClient } from "@aws-sdk/client-lex-runtime-service";

const REGION = "REGION";
const IDENTITY_POOL_ID = "IDENTITY_POOL_ID"; // An Amazon Cognito Identity Pool ID.

// Create an Amazon Lex service client object.
const lexClient = new LexRuntimeServiceClient({
  region: REGION,
  credentials: fromCognitoIdentityPool({
    client: new CognitoIdentityClient({ region: REGION }),
    identityPoolId: IDENTITY_POOL_ID,
  }),
});
```

```
export { lexClient };
```

```
import { CognitoIdentityClient } from "@aws-sdk/client-cognito-identity";
import { fromCognitoIdentityPool } from "@aws-sdk/credential-provider-cognito-
identity";
import { TranslateClient } from "@aws-sdk/client-translate";

const REGION = "REGION";
const IDENTITY_POOL_ID = "IDENTITY_POOL_ID"; // An Amazon Cognito Identity Pool ID.

// Create an Amazon Translate service client object.
const translateClient = new TranslateClient({
  region: REGION,
  credentials: fromCognitoIdentityPool({
    client: new CognitoIdentityClient({ region: REGION }),
    identityPoolId: IDENTITY_POOL_ID,
  }),
});

export { translateClient };
```

此代码可从此处获得 GitHub。 。

接下来，创建一个 `index.js` 文件，并将以下代码粘贴到文件中。

将 *BOT_ALIAS* 和 *BOT_NAME* 分别替换为您的 Amazon Lex 机器人的别名和名称，将 *USER_ID* 替换为用户 ID。`createResponse` 异步函数将执行以下操作：

- 将用户输入的文本导入浏览器，然后使用 Amazon Comprehend 来确定其语言代码。
- 获取语言代码并使用 Amazon Translate 将文本翻译成英文。
- 获取翻译后的文本并使用 Amazon Lex 生成响应。
- 将响应发布到浏览器页面。

```
import { DetectDominantLanguageCommand } from "@aws-sdk/client-comprehend";
import { TranslateTextCommand } from "@aws-sdk/client-translate";
import { PostTextCommand } from "@aws-sdk/client-lex-runtime-service";
import { lexClient } from "./libs/lexClient.js";
import { translateClient } from "./libs/translateClient.js";
import { comprehendClient } from "./libs/comprehendClient.js";
```

```
var g_text = "";
// Set the focus to the input box.
document.getElementById("wisdom").focus();

function showRequest() {
  var conversationDiv = document.getElementById("conversation");
  var requestPara = document.createElement("P");
  requestPara.className = "userRequest";
  requestPara.appendChild(document.createTextNode(g_text));
  conversationDiv.appendChild(requestPara);
  conversationDiv.scrollTop = conversationDiv.scrollHeight;
}

function showResponse(lexResponse) {
  var conversationDiv = document.getElementById("conversation");
  var responsePara = document.createElement("P");
  responsePara.className = "lexResponse";

  var lexTextResponse = lexResponse;

  responsePara.appendChild(document.createTextNode(lexTextResponse));
  responsePara.appendChild(document.createElement("br"));
  conversationDiv.appendChild(responsePara);
  conversationDiv.scrollTop = conversationDiv.scrollHeight;
}

function handletext(text) {
  g_text = text;
  var xhr = new XMLHttpRequest();
  xhr.addEventListener("load", loadNewItems, false);
  xhr.open("POST", "../text", true); // A Spring MVC controller
  xhr.setRequestHeader("Content-type", "application/x-www-form-urlencoded"); //
necessary
  xhr.send("text=" + text);
}

function loadNewItems() {
  showRequest();

  // Re-enable input.
  var wisdomText = document.getElementById("wisdom");
  wisdomText.value = "";
  wisdomText.locked = false;
}
```

```
// Respond to user's input.
const createResponse = async () => {
  // Confirm there is text to submit.
  var wisdomText = document.getElementById("wisdom");
  if (wisdomText && wisdomText.value && wisdomText.value.trim().length > 0) {
    // Disable input to show it is being sent.
    var wisdom = wisdomText.value.trim();
    wisdomText.value = "...";
    wisdomText.locked = true;
    handletext(wisdom);

    const comprehendParams = {
      Text: wisdom,
    };
    try {
      const data = await comprehendClient.send(
        new DetectDominantLanguageCommand(comprehendParams)
      );
      console.log(
        "Success. The language code is: ",
        data.Languages[0].LanguageCode
      );
      const translateParams = {
        SourceLanguageCode: data.Languages[0].LanguageCode,
        TargetLanguageCode: "en", // For example, "en" for English.
        Text: wisdom,
      };
      try {
        const data = await translateClient.send(
          new TranslateTextCommand(translateParams)
        );
        console.log("Success. Translated text: ", data.TranslatedText);
        const lexParams = {
          botName: "BookTrip",
          botAlias: "mynewalias",
          inputText: data.TranslatedText,
          userId: "chatbot", // For example, 'chatbot-demo'.
        };
        try {
          const data = await lexClient.send(new PostTextCommand(lexParams));
          console.log("Success. Response is: ", data.message);
          var msg = data.message;
          showResponse(msg);
```

```
        } catch (err) {
          console.log("Error responding to message. ", err);
        }
      } catch (err) {
        console.log("Error translating text. ", err);
      }
    } catch (err) {
      console.log("Error identifying language. ", err);
    }
  }
};
// Make the function available to the browser.
window.createResponse = createResponse;
```

此代码可从此处获得 GitHub。 。

现在使用 Webpack 将 `index.js` 和 Amazon SDK for JavaScript 模块捆绑到一个文件 `main.js` 中。

1. 如果您尚未安装 Webpack，请按照本示例中的先决条件部分进行安装。

> **Note**
>
> 有关 webpack 的信息，请参阅将应用程序与 webpack 捆绑在一起。

2. 在命令行中运行以下命令，将本示例的捆绑到名 JavaScript 为的文件中`main.js`：

```
webpack index.js --mode development --target web --devtool false -o main.js
```

后续步骤

恭喜您！您创建了一个 Node.js 应用程序，该应用程序使用 Amazon Lex 来创建交互式用户体验。如本教程开头所述，请务必在学习本教程时终止您创建的所有资源，以确保系统不会向您收费。您可以通过删除在本教程的创建 Amazon 资源 主题中创建的 Amazon CloudFormation 堆栈来实现此目的，方法如下所示：

1. 打开Amazon CloudFormation控制台。
2. 在堆栈页面上，选择堆栈。
3. 选择删除。

有关更多 Amazon 跨服务示例，请参阅 Amazon SDK for JavaScript 跨服务示例。

创建示例消息收发应用程序

您可以使用 Amazon SDK for JavaScript 和 Amazon Simple Queue Service (Amazon SQS) 创建用于发送和检索消息的 Amazon 应用程序。消息存储在先进先出 (FIFO) 队列中，该队列可确保消息的顺序一致。例如，存储在队列中的第一条消息是从队列中读取的第一条消息。

> **Note**
> 有关 Amazon SQS 的更多信息，请参阅什么是 Amazon Simple Queue Service？

在本教程中，您将创建一个名为 Amazon Messaging 的 Node.js 应用程序。

完成费用：本文档中包含的 Amazon 服务包含在 Amazon 免费套餐中。

注意：在学习本教程时，请务必终止您创建的所有资源，以确保系统不会向您收费。

构建应用程序：

1. 先决条件
2. 预置资源
3. 了解工作流
4. 创建 HTML
5. 创建浏览器脚本
6. 后续步骤

先决条件

要设置和运行此示例，您必须先完成以下任务：

- 设置项目环境以运行这些 Node TypeScript 示例，并安装所需的模块Amazon SDK for JavaScript和第三方模块。按照上的说明进行操作 GitHub。
- 使用用户凭证创建共享配置文件。有关提供共享凭证文件的更多信息，请参阅《Amazon SDK 和工具参考指南》中的共享配置和凭证文件。

> **Important**
>
> 此示例使用 ECMAScript6 (ES6)。这需要使用 Node.js 版本 13.x 或更高版本。要下载并安装最新版本的 Node.js，请参阅 Node.js 下载。
> 但是，如果您更喜欢使用 CommonJS 语法，请参阅 JavaScript ES6/CommonJS 语法。

创建 Amazon 资源

本教程要求具有以下资源。

- 一个未经身份验证的 IAM 角色，具有 Amazon SQS 权限。
- 一个名为 Message.fifo 的 FIFO Amazon SQS 队列 - 有关创建队列的信息，请参阅创建 Amazon SQS 队列。

您可以手动创建这些资源，但我们建议使用本教程中所述的 Amazon CloudFormation (Amazon CloudFormation) 预置这些资源。

> **Note**
>
> Amazon CloudFormation 是一个软件开发框架，使您能够定义云应用程序资源。有关更多信息，请参阅 Amazon CloudFormation 用户指南。

使用 Amazon CloudFormation 创建 Amazon 资源

Amazon CloudFormation 让您能够以可预测、可重复的方式创建和预置 Amazon 基础设施部署。有关 Amazon CloudFormation 的更多信息，请参阅 Amazon CloudFormation 用户指南。

使用 Amazon CLI 创建 Amazon CloudFormation 堆栈：

1. 按照 Amazon CLI 用户指南中的说明安装和配置 Amazon CLI。
2. 在项目文件夹的根目录`setup.yaml`中创建一个名为的文件，然后将此处的内容复制 GitHub到该文件中。

Note

该Amazon CloudFormation模板是使用此处Amazon CDK提供的模板生成的 GitHub。有关 Amazon CDK 的更多信息，请参阅 Amazon Cloud Development Kit (Amazon CDK) 开发人员指南。

3. 从命令行运行以下命令，将 *STACK_NAME* 替换为堆栈的唯一名称。

> Important
>
> 在一个 Amazon 区域和一个 Amazon 账户中，堆栈名称必须唯一。您最多可指定 128 个字符，支持数字和连字符。

```
aws cloudformation create-stack --stack-name STACK_NAME --template-body file://
setup.yaml --capabilities CAPABILITY_IAM
```

有关 `create-stack` 命令参数的更多信息，请参阅 Amazon CLI 命令参考指南和 Amazon CloudFormation 用户指南。

要查看创建的资源，请在 Amazon 管理控制台中打开 Amazon CloudFormation，选择堆栈，然后选择资源选项卡。

了解 Amazon 消息收发应用程序

要向 SQS 队列发送消息，请将消息输入到应用程序中并选择“发送”。

发送消息后，应用程序会显示该消息。

您可以选择清除，从 Amazon SQS 队列中清除消息。这会导致队列为空，并且应用程序中不会显示任何消息。

以下内容描述了应用程序处理消息的方式：

- 用户选择自己的名字并输入消息，然后提交消息，这将启动 `pushMessage` 函数。
- `pushMessage` 检索 Amazon SQS 队列 URL，然后向 Amazon SQS 队列发送一条带有唯一消息 ID 值 (GUID)、消息文本和用户的消息。

- pushMessage 从 Amazon SQS 队列中检索消息，提取每条消息的用户和消息，然后显示消息。
- 用户可以清除消息，从而将消息从 Amazon SQS 队列和用户界面中删除。

创建 HTML 页面

现在，您可以创建应用程序的图形用户界面 (GUI) 所需的 HTML 文件。创建一个名为 index.html的文件。将以下代码复制并粘贴到 index.html。此 HTML 引用 main.js。这是 index.js 的捆绑版本，其中包含所需的 Amazon SDK for JavaScript 模块。

```
<!doctype html>
<html
  xmlns:th="http://www.thymeleaf.org"
  xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity3"
>
  <head>
    <meta charset="utf-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <link rel="icon" href="./images/favicon.ico" />
    <link
      rel="stylesheet"
      href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
    />
    <link rel="stylesheet" href="./css/styles.css" />
    <script src="https://code.jquery.com/jquery-1.12.4.min.js"></script>
    <script src="https://code.jquery.com/ui/1.11.4/jquery-ui.min.js"></script>
    <script src="./js/main.js"></script>
    <style>
      .messageelement {
        margin: auto;
        border: 2px solid #dedede;
        background-color: #d7d1d0;
        border-radius: 5px;
        max-width: 800px;
        padding: 10px;
        margin: 10px 0;
      }

      .messageelement::after {
        content: "";
        clear: both;
        display: table;
```

```
      }

      .messageelement img {
        float: left;
        max-width: 60px;
        width: 100%;
        margin-right: 20px;
        border-radius: 50%;
      }

      .messageelement img.right {
        float: right;
        margin-left: 20px;
        margin-right: 0;
      }
    </style>
  </head>
  <body>
    <div class="container">
      <h2>AWS Sample Messaging Application</h2>
      <div id="messages"></div>

      <div class="input-group mb-3">
        <div class="input-group-prepend">
          <span class="input-group-text" id="basic-addon1">Sender:</span>
        </div>
        <select name="cars" id="username">
          <option value="Scott">Brian</option>
          <option value="Tricia">Tricia</option>
        </select>
      </div>

      <div class="input-group">
        <div class="input-group-prepend">
          <span class="input-group-text">Message:</span>
        </div>
        <textarea
          class="form-control"
          id="textarea"
          aria-label="With textarea"
        ></textarea>
        <button
          type="button"
          onclick="pushMessage()"
```

```
          id="send"
          class="btn btn-success"
        >
          Send
        </button>
        <button
          type="button"
          onclick="purge()"
          id="refresh"
          class="btn btn-success"
        >
          Purge
        </button>
      </div>
      <!-- All of these child items are hidden and only displayed in a FancyBox
 ------------------------------------------------------->
      <div id="hide" style="display: none">
        <div id="base" class="messageelement">
          <img
            src="../public/images/av2.png"
            alt="Avatar"
            class="right"
            style="width: 100%"
          />
          <p id="text">Excellent! So, what do you want to do today?</p>
          <span class="time-right">11:02</span>
        </div>
      </div>
    </div>
  </body>
</html>
```

此代码也可以在此处找到 GitHub。

创建浏览器脚本

在本主题中，您将为应用程序创建浏览器脚本。创建浏览器脚本后，可以将其捆绑到一个名为 `main.js` 的文件中，如捆绑起来 JavaScript 中所述。

创建一个名为 `index.js` 的文件。从此处复制代码并将其粘贴 GitHub到其中。

以下部分对此代码进行了说明：

1. 配置

2. populateChat
3. 推送消息
4. purge

配置

首先，创建一个 `libs` 目录，然后通过创建名为 `sqsClient.js` 的文件来创建所需的 Amazon SQS 客户端对象。替换每个对象中的 *`REGION`* 和 *`IDENTITY_POOL_ID`*。

> **Note**
> 使用您在创建 Amazon 资源 中创建的 Amazon Cognito 身份池的 ID。

```
import { CognitoIdentityClient } from "@aws-sdk/client-cognito-identity";
import { fromCognitoIdentityPool } from "@aws-sdk/credential-providers";
import {SQSClient} from "@aws-sdk/client-sqs";
const REGION = "REGION"; //e.g. "us-east-1"
const IdentityPoolId = "IDENTITY_POOL_ID";
const sqsClient = new SQSClient({
  region: REGION,
  credentials: fromCognitoIdentityPool({
    client: new CognitoIdentityClient({ region: REGION }),
    identityPoolId: IdentityPoolId
  }),
});
```

在 `index.js` 中，导入所需的 Amazon SDK for JavaScript 模块和命令。将 *`SQS_QUEUE_NAME`* 替换为您在创建 Amazon 资源 中创建的 Amazon SQS 队列的名称。

```
import {
  GetQueueUrlCommand,
  SendMessageCommand,
  ReceiveMessageCommand,
  PurgeQueueCommand,
} from "@aws-sdk/client-sqs";
import { sqsClient } from "./libs/sqsClient.js";
```

```
const QueueName = "SQS_QUEUE_NAME"; // The Amazon SQS queue name, which must end
 in .fifo for this example.
```

populateChat

`populateChat` 函数 onload 会自动检索 Amazon SQS 队列的 URL，检索队列中的所有消息并将其显示出来。

```
$(function () {
  populateChat();
});

const populateChat = async () => {
  try {
    // Set the Amazon SQS Queue parameters.
    const queueParams = {
      QueueName: QueueName,
      Attributes: {
        DelaySeconds: "60",
        MessageRetentionPeriod: "86400",
      },
    };
    // Get the Amazon SQS Queue URL.
    const data = await sqsClient.send(new GetQueueUrlCommand(queueParams));
    console.log("Success. The URL of the SQS Queue is: ", data.QueueUrl);
    // Set the parameters for retrieving the messages in the Amazon SQS Queue.
    var getMessageParams = {
      QueueUrl: data.QueueUrl,
      MaxNumberOfMessages: 10,
      MessageAttributeNames: ["All"],
      VisibilityTimeout: 20,
      WaitTimeSeconds: 20,
    };
    try {
      // Retrieve the messages from the Amazon SQS Queue.
      const data = await sqsClient.send(
        new ReceiveMessageCommand(getMessageParams)
      );
      console.log("Successfully retrieved messages", data.Messages);

      // Loop through messages for user and message body.
      var i;
```

```
      for (i = 0; i < data.Messages.length; i++) {
        const name = data.Messages[i].MessageAttributes.Name.StringValue;
        const body = data.Messages[i].Body;
        // Create the HTML for the message.
        var userText = body + "<br><br><b>" + name;
        var myTextNode = $("#base").clone();
        myTextNode.text(userText);
        var image_url;
        var n = name.localeCompare("Scott");
        if (n == 0) image_url = "./images/av1.png";
        else image_url = "./images/av2.png";
        var images_div =
          '<img src="' +
          image_url +
          '" alt="Avatar" class="right" style=""width:100%;"">';
        myTextNode.html(userText);
        myTextNode.append(images_div);

        // Add the message to the GUI.
        $("#messages").append(myTextNode);
      }
    } catch (err) {
      console.log("Error loading messages: ", err);
    }
  } catch (err) {
    console.log("Error retrieving SQS queue URL: ", err);
  }
};
```

推送消息

用户选择自己的名字并输入消息，然后提交消息，这将启动 pushMessage 函数。pushMessage 检索 Amazon SQS 队列 URL，然后向 Amazon SQS 队列发送一条带有唯一消息 ID 值 (GUID)、消息文本和用户的消息。然后，它会从 Amazon SQS 队列中检索所有消息并将其显示出来。

```
const pushMessage = async () => {
  // Get and convert user and message input.
  var user = $("#username").val();
  var message = $("#textarea").val();

  // Create random deduplication ID.
  var dt = new Date().getTime();
  var uuid = "xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx".replace(/[xy]/g, function (
```

```
    c
  ) {
    var r = (dt + Math.random() * 16) % 16 | 0;
    dt = Math.floor(dt / 16);
    return (c == "x" ? r : (r & 0x3) | 0x8).toString(16);
  });

  try {
    // Set the Amazon SQS Queue parameters.
    const queueParams = {
      QueueName: QueueName,
      Attributes: {
        DelaySeconds: "60",
        MessageRetentionPeriod: "86400",
      },
    };
    const data = await sqsClient.send(new GetQueueUrlCommand(queueParams));
    console.log("Success. The URL of the SQS Queue is: ", data.QueueUrl);
    // Set the parameters for the message.
    var messageParams = {
      MessageAttributes: {
        Name: {
          DataType: "String",
          StringValue: user,
        },
      },
      MessageBody: message,
      MessageDeduplicationId: uuid,
      MessageGroupId: "GroupA",
      QueueUrl: data.QueueUrl,
    };
    const result = await sqsClient.send(new SendMessageCommand(messageParams));
    console.log("Success", result.MessageId);

    // Set the parameters for retrieving all messages in the SQS queue.
    var getMessageParams = {
      QueueUrl: data.QueueUrl,
      MaxNumberOfMessages: 10,
      MessageAttributeNames: ["All"],
      VisibilityTimeout: 20,
      WaitTimeSeconds: 20,
    };

    // Retrieve messages from SQS Queue.
```

```
    const final = await sqsClient.send(
      new ReceiveMessageCommand(getMessageParams)
    );
    console.log("Successfully retrieved", final.Messages);
    $("#messages").empty();
    // Loop through messages for user and message body.
    var i;
    for (i = 0; i < final.Messages.length; i++) {
      const name = final.Messages[i].MessageAttributes.Name.StringValue;
      const body = final.Messages[i].Body;
      // Create the HTML for the message.
      var userText = body + "<br><br><b>" + name;
      var myTextNode = $("#base").clone();
      myTextNode.text(userText);
      var image_url;
      var n = name.localeCompare("Scott");
      if (n == 0) image_url = "./images/av1.png";
      else image_url = "./images/av2.png";
      var images_div =
        '<img src="' +
        image_url +
        '" alt="Avatar" class="right" style=""width:100%;"">';
      myTextNode.html(userText);
      myTextNode.append(images_div);
      // Add the HTML to the GUI.
      $("#messages").append(myTextNode);
    }
  } catch (err) {
    console.log("Error", err);
  }
};
// Make the function available to the browser window.
window.pushMessage = pushMessage;
```

清除消息

purge 从 Amazon SQS 队列和用户界面中删除消息。

```
// Delete the message from the Amazon SQS queue.
const purge = async () => {
  try {
    // Set the Amazon SQS Queue parameters.
    const queueParams = {
      QueueName: QueueName,
```

```
      Attributes: {
        DelaySeconds: "60",
        MessageRetentionPeriod: "86400",
      },
    };
    // Get the Amazon SQS Queue URL.
    const data = await sqsClient.send(new GetQueueUrlCommand(queueParams));
    console.log("Success", data.QueueUrl);
    // Delete all the messages in the Amazon SQS Queue.
    const result = await sqsClient.send(
      new PurgeQueueCommand({ QueueUrl: data.QueueUrl })
    );
    // Delete all the messages from the GUI.
    $("#messages").empty();
    console.log("Success. All messages deleted.", data);
  } catch (err) {
    console.log("Error", err);
  }
};

// Make the function available to the browser window.
window.purge = purge;
```

捆绑起来 JavaScript

此完整的浏览器脚本代码可从此处获得。 GitHub 。

现在使用 Webpack 将 `index.js` 和 Amazon SDK for JavaScript 模块捆绑到一个文件 `main.js` 中。

1. 如果您尚未安装 Webpack，请按照本示例中的先决条件部分进行安装。

   > **Note**
   >
   > 有关 webpack 的信息，请参阅将应用程序与 webpack 捆绑在一起。

2. 在命令行中运行以下命令，将本示例的捆绑到名 JavaScript 为的文件中`<index.js>`：

   ```
   webpack index.js --mode development --target web --devtool false -o main.js
   ```

后续步骤

恭喜您！您已经创建并部署了使用 Amazon SQS 的 Amazon 消息收发应用程序。如本教程开头所述，请务必在学习本教程时终止您创建的所有资源，以确保系统不再向您收费。您可以通过删除在本教程的创建 Amazon 资源 主题中创建的 Amazon CloudFormation 堆栈来实现此目的，方法如下所示：

1. 打开 Amazon 管理控制台中的 Amazon CloudFormation。
2. 打开堆栈页面，然后选择堆栈。
3. 选择删除。

Amazon Cloud9 搭配使用 Amazon SDK for JavaScript

您只需 Amazon Cloud9 使用浏览器即可在 Amazon SDK for JavaScript 浏览器 JavaScript 中编写和运行您的代码，以及编写、运行和调试您的 Node.js 代码。 Amazon Cloud9 包括代码编辑器和终端等工具，以及用于 Node.js 代码的调试器。

由于 Amazon Cloud9 IDE 是基于云的，因此您可以在办公室、家中或任何地方使用联网的计算机处理项目。有关的一般信息 Amazon Cloud9，请参阅《Amazon Cloud9 用户指南》。

以下步骤描述了如何使用的 SDK Amazon Cloud9 进行设置 JavaScript。

目录

- 第 1 步：设置要使用的 Amazon 账户 Amazon Cloud9
- 第 2 步：设置 Amazon Cloud9 开发环境
- 步骤 3：设置 SDK JavaScript
  - 为 Node.js 设置软件开发 JavaScript 工具包
  - 在浏览器 JavaScript 中设置 SDK
- 步骤 4：下载示例代码
- 步骤 5：运行并调试示例代码

第 1 步：设置要使用的 Amazon 账户 Amazon Cloud9

以拥有您 Amazon 账户访问权限 Amazon Cloud9 的 Amazon Identity and Access Management (IAM) 实体（例如，IAM 用户）身份登录 Amazon Cloud9 控制台，即可开始使用。 Amazon Cloud9

要在您的 Amazon 账户中设置 IAM 实体以进行访问 Amazon Cloud9和登录 Amazon Cloud9 控制台，请参阅Amazon Cloud9 用户指南 Amazon Cloud9中的团队设置。

第 2 步：设置 Amazon Cloud9 开发环境

登录 Amazon Cloud9 控制台后，使用控制台创建 Amazon Cloud9 开发环境。创建环境后， Amazon Cloud9 打开该环境的 IDE。

请参阅《Amazon Cloud9 用户指南》中的在 Amazon Cloud9中创建环境，了解详细信息。

Note

在控制台中首次创建环境之后，我们建议您选择创建新的环境实例 (EC2)。此选项指示 Amazon Cloud9 创建环境，启动 Amazon EC2 实例，然后将新实例连接到新环境。这是开始使用的最快方法 Amazon Cloud9。

步骤 3：设置 SDK JavaScript

为您的开发环境 Amazon Cloud9 打开 IDE 后，请按照以下一个或两个过程使用 IDE JavaScript 在您的环境中为其设置 SDK。

为 Node.js 设置软件开发 JavaScript 工具包

1. 如果终端未在 IDE 中打开，请打开它。为此，请在 IDE 中的菜单栏上，选择窗口、新终端。
2. 运行以下命令npm以用于安装适用的 SDK `Cloud9` 客户端 JavaScript。

```
npm install @aws-sdk/client-cloud9
```

如果 IDE 无法找到 npm，请按照以下顺序一次运行一个命令来安装 npm。（这些命令假定您已在本主题的前面选择了创建新的环境实例 (EC2) 选项。）

Warning

Amazon 不控制以下代码。在运行之前，请务必验证其真实性和完整性。有关此代码的更多信息可以在 nvm（节点版本管理器）GitHub 存储库中找到。

```
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.34.0/install.sh | bash #
 Download and install Node Version Manager (nvm).
. ~/.bashrc                                                                       #
 Activate nvm.
nvm install node                                                                  #
 Use nvm to install npm (and Node.js at the same time).
```

在浏览器 JavaScript 中设置 SDK

要在 HTML 页面 JavaScript 中使用适用的 SDK， WebPack 请使用将所需的客户端模块和所有必需的 JavaScript 函数捆绑到一个 JavaScript 文件中，然后将其添加到 HTML 页面<head>的脚本标签中。例如：

```
<script src=./main.js></script>
```

> **Note**
>
> 有关 Webpack 的更多信息，请参阅将应用程序与 webpack 捆绑在一起

步骤 4：下载示例代码

使用您在上一步中打开的终端将 SDK 的示例代码下载 JavaScript 到 Amazon Cloud9 开发环境中。（如果未在 IDE 中打开终端，请通过在 IDE 中的菜单栏上选择窗口、新终端来打开它。）

要下载示例代码，请运行以下命令。此命令将官方 Amazon SDK 文档中使用的所有代码示例的副本下载到您环境的根目录中。

```
git clone https://github.com/awsdocs/aws-doc-sdk-examples.git
```

要查找的 SDK 的代码示例 JavaScript，请使用环境窗口打开*ENVIRONMENT_NAME*`\aws-doc-sdk-examples\javascriptv3\example_code/src`，其中 ENVIRONMEN *T_NAME ### Amazon Cloud9 ####*的名称。

要了解如何使用这些代码示例和其他代码示例，请参阅 SDK 以获取 JavaScript 代码示例。

步骤 5：运行并调试示例代码

要在 Amazon Cloud9 开发环境中运行代码，请参阅Amazon Cloud9 用户指南中的运行代码。

要调试 Node.js 代码，请参阅《Amazon Cloud9 用户指南》中的调试代码。

适用于 JavaScript (v3) 代码示例的 SDK

本主题中的代码示例向您展示了如何将 Amazon SDK for JavaScript (v3) 与 Amazon。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景 是展示如何通过在同一服务中调用多个函数来完成特定任务的代码示例。

跨服务示例是指跨多个 Amazon Web Services工作的示例应用程序。

示例

- 使用适用于 JavaScript (v3) 的 SDK 的操作和场景
- 使用适用于 JavaScript (v3) 的 SDK 的跨服务示例

使用适用于 JavaScript (v3) 的 SDK 的操作和场景

以下代码示例演示如何使用带的 Amazon SDK for JavaScript (v3) 来执行操作和实现常见场景 Amazon Web Services。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景是指显示如何通过在同一服务中调用多个函数来完成特定任务的代码示例。

服务

- 使用适用于 JavaScript (v3) 的 SDK 的 Auto Scaling 示例
- 使用适用于 JavaScript (v3) 的 SDK 的 Amazon Bedrock 示例
- 使用适用于 JavaScript (v3) 的 SDK 的亚马逊 Bedrock 运行时示例
- 使用适用于 JavaScript (v3) 的软件开发工具包的 Amazon Bedrock 代理示例
- 使用适用于 JavaScript (v3) 的 SDK 的 Amazon Bedrock 运行时代理示例
- CloudWatch 使用适用于 JavaScript (v3) 的 SDK 的示例
- CloudWatch 使用适用于 JavaScript (v3) 的 SDK 的事件示例
- CloudWatch 使用适用于 JavaScript (v3) 的 SDK 记录示例
- CodeBuild 使用适用于 JavaScript (v3) 的 SDK 的示例
- 使用适用于 JavaScript (v3) 的软件开发工具包的 Amazon Cognito 身份提供商示例

- 使用适用于 (v3) 的 SDK JavaScript 的 DynamoDB 示例
- 使用适用于 JavaScript (v3) 的软件开发工具包的 Amazon EC2 示例
- Elastic Load Balancing 示例，使用适用于 JavaScript (v3) 的
- EventBridge 使用适用于 JavaScript (v3) 的 SDK 的示例
- Amazon Glue 使用适用于 JavaScript (v3) 的 SDK 的示例
- HealthImaging 使用适用于 JavaScript (v3) 的 SDK 的示例
- 使用适用于 JavaScript (v3) 的开发工具包的 IAM 示例
- 使用适用于 JavaScript (v3) 的软件开发工具包的 Lambda 示例
- Amazon 使用适用于 JavaScript (v3) 的软件开发工具包对示例进行个性化设置
- Amazon 使用适用于 JavaScript (v3) 的软件开发工具包对事件进行个性化设置示例
- Amazon 使用适用于 JavaScript (v3) 的软件开发工具包对运行时进行个性化示例
- 使用适用于 JavaScript (v3) 的软件开发工具包的亚马逊 Pinpoint 示例
- 使用适用于 JavaScript (v3) 的软件开发工具包的亚马逊 Redshift 示例
- 使用适用于 JavaScript (v3) 的软件开发工具包的 Amazon S3 示例
- 使用 JavaScript (v3) 软件开发工具包的 S3 Glacier 示例
- SageMaker 使用适用于 JavaScript (v3) 的 SDK 的示例
- 使用适用于 JavaScript (v3) 的 SDK 的 Secrets Manager 示例
- 使用适用于 JavaScript (v3) 的软件开发工具包的 Amazon SES 示例
- 使用适用于 JavaScript (v3) 的软件开发工具包的亚马逊 SNS 示例
- 使用适用于 JavaScript (v3) 的软件开发工具包的亚马逊 SQS 示例
- 使用 JavaScript (v3) 软件开发工具包的 Step Functions 示例
- Amazon STS 使用适用于 JavaScript (v3) 的 SDK 的示例
- Amazon Web Services Support 使用适用于 JavaScript (v3) 的 SDK 的示例
- 使用适用于 JavaScript (v3) 的软件开发工具包的 Amazon Transcribe 示例

使用适用于 JavaScript (v3) 的 SDK 的 Auto Scaling 示例

以下代码示例向您展示了如何使用带有 Auto Scaling 的 Amazon SDK for JavaScript (v3) 来执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景 是展示如何通过在同一服务中调用多个函数来完成特定任务的代码示例。

每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关如何在上下文中设置和运行代码的说明。

主题

- 操作
- 场景

操作

将 ELB 目标组附加到 Auto Scaling 组

以下代码示例演示了如何将 ELB 目标组附加到 Auto Scaling 组。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
const client = new AutoScalingClient({});
await client.send(
  new AttachLoadBalancerTargetGroupsCommand({
    AutoScalingGroupName: NAMES.autoScalingGroupName,
    TargetGroupARNs: [state.targetGroupArn],
  }),
);
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考AttachLoadBalancerTargetGroups中的。

场景

构建和管理弹性服务

以下代码示例演示了如何创建可返回书籍、电影和歌曲推荐的负载均衡的 Web 服务。该示例演示服务如何响应故障，以及如何重组服务以提高故障发生时的弹性。

- 使用 Amazon EC2 Auto Scaling 组根据启动模板创建 Amazon Elastic Compute Cloud（Amazon EC2）实例，并将实例数量保持在指定范围内。
- 使用弹性负载均衡处理和分发 HTTP 请求。
- 监控自动扩缩组中实例的运行状况，并仅将请求转发到运行状况良好的实例。
- 在每个 EC2 实例上运行 Python Web 服务器以处理 HTTP 请求。Web 服务器以建议和运行状况检查作为响应。
- 使用 Amazon DynamoDB 表模拟推荐服务。
- 通过更新 Amazon Systems Manager 参数来控制 Web 服务器对请求和运行状况检查的响应。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

在命令提示符中运行交互式场景。

```
#!/usr/bin/env node
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0


import {
  Scenario,
  parseScenarioArgs,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

/**
 * The workflow steps are split into three stages:
 *   - deploy
```

```
 *   - demo
 *   - destroy
 *
 * Each of these stages has a corresponding file prefixed with steps-*.
 */
import { deploySteps } from "./steps-deploy.js";
import { demoSteps } from "./steps-demo.js";
import { destroySteps } from "./steps-destroy.js";

/**
 * The context is passed to every scenario. Scenario steps
 * will modify the context.
 */
const context = {};

/**
 * Three Scenarios are created for the workflow. A Scenario is an orchestration
 class
 * that simplifies running a series of steps.
 */
export const scenarios = {
  // Deploys all resources necessary for the workflow.
  deploy: new Scenario("Resilient Workflow - Deploy", deploySteps, context),
  // Demonstrates how a fragile web service can be made more resilient.
  demo: new Scenario("Resilient Workflow - Demo", demoSteps, context),
  // Destroys the resources created for the workflow.
  destroy: new Scenario("Resilient Workflow - Destroy", destroySteps, context),
};

// Call function if run directly
import { fileURLToPath } from "url";

if (process.argv[1] === fileURLToPath(import.meta.url)) {
  parseScenarioArgs(scenarios);
}
```

创建部署所有资源的步骤。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { join } from "node:path";
import { readFileSync, writeFileSync } from "node:fs";
```

```
import axios from "axios";

import {
  BatchWriteItemCommand,
  CreateTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";
import {
  EC2Client,
  CreateKeyPairCommand,
  CreateLaunchTemplateCommand,
  DescribeAvailabilityZonesCommand,
  DescribeVpcsCommand,
  DescribeSubnetsCommand,
  DescribeSecurityGroupsCommand,
  AuthorizeSecurityGroupIngressCommand,
} from "@aws-sdk/client-ec2";
import {
  IAMClient,
  CreatePolicyCommand,
  CreateRoleCommand,
  CreateInstanceProfileCommand,
  AddRoleToInstanceProfileCommand,
  AttachRolePolicyCommand,
  waitUntilInstanceProfileExists,
} from "@aws-sdk/client-iam";
import { SSMClient, GetParameterCommand } from "@aws-sdk/client-ssm";
import {
  CreateAutoScalingGroupCommand,
  AutoScalingClient,
  AttachLoadBalancerTargetGroupsCommand,
} from "@aws-sdk/client-auto-scaling";
import {
  CreateListenerCommand,
  CreateLoadBalancerCommand,
  CreateTargetGroupCommand,
  ElasticLoadBalancingV2Client,
  waitUntilLoadBalancerAvailable,
} from "@aws-sdk/client-elastic-load-balancing-v2";

import {
  ScenarioOutput,
  ScenarioInput,
```

```
  ScenarioAction,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES, RESOURCES_PATH, ROOT } from "./constants.js";
import { initParamsSteps } from "./steps-reset-params.js";

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const deploySteps = [
  new ScenarioOutput("introduction", MESSAGES.introduction, { header: true }),
  new ScenarioInput("confirmDeployment", MESSAGES.confirmDeployment, {
    type: "confirm",
  }),
  new ScenarioAction(
    "handleConfirmDeployment",
    (c) => c.confirmDeployment === false && process.exit(),
  ),
  new ScenarioOutput(
    "creatingTable",
    MESSAGES.creatingTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
  new ScenarioAction("createTable", async () => {
    const client = new DynamoDBClient({});
    await client.send(
      new CreateTableCommand({
        TableName: NAMES.tableName,
        ProvisionedThroughput: {
          ReadCapacityUnits: 5,
          WriteCapacityUnits: 5,
        },
        AttributeDefinitions: [
          {
            AttributeName: "MediaType",
            AttributeType: "S",
          },
          {
            AttributeName: "ItemId",
            AttributeType: "N",
          },
        ],
        KeySchema: [
          {
```

```
            AttributeName: "MediaType",
            KeyType: "HASH",
          },
          {
            AttributeName: "ItemId",
            KeyType: "RANGE",
          },
        ],
      }),
    );
    await waitUntilTableExists({ client }, { TableName: NAMES.tableName });
  }),
  new ScenarioOutput(
    "createdTable",
    MESSAGES.createdTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
  new ScenarioOutput(
    "populatingTable",
    MESSAGES.populatingTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
  new ScenarioAction("populateTable", () => {
    const client = new DynamoDBClient({});
    /**
     * @type {{ default: import("@aws-sdk/client-dynamodb").PutRequest['Item'][] }}
     */
    const recommendations = JSON.parse(
      readFileSync(join(RESOURCES_PATH, "recommendations.json")),
    );

    return client.send(
      new BatchWriteItemCommand({
        RequestItems: {
          [NAMES.tableName]: recommendations.map((item) => ({
            PutRequest: { Item: item },
          })),
        },
      }),
    );
  }),
  new ScenarioOutput(
    "populatedTable",
    MESSAGES.populatedTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
  new ScenarioOutput(
```

```
    "creatingKeyPair",
    MESSAGES.creatingKeyPair.replace("${KEY_PAIR_NAME}", NAMES.keyPairName),
  ),
  new ScenarioAction("createKeyPair", async () => {
    const client = new EC2Client({});
    const { KeyMaterial } = await client.send(
      new CreateKeyPairCommand({
        KeyName: NAMES.keyPairName,
      }),
    );

    writeFileSync(`${NAMES.keyPairName}.pem`, KeyMaterial, { mode: 0o600 });
  }),
  new ScenarioOutput(
    "createdKeyPair",
    MESSAGES.createdKeyPair.replace("${KEY_PAIR_NAME}", NAMES.keyPairName),
  ),
  new ScenarioOutput(
    "creatingInstancePolicy",
    MESSAGES.creatingInstancePolicy.replace(
      "${INSTANCE_POLICY_NAME}",
      NAMES.instancePolicyName,
    ),
  ),
  new ScenarioAction("createInstancePolicy", async (state) => {
    const client = new IAMClient({});
    const {
      Policy: { Arn },
    } = await client.send(
      new CreatePolicyCommand({
        PolicyName: NAMES.instancePolicyName,
        PolicyDocument: readFileSync(
          join(RESOURCES_PATH, "instance_policy.json"),
        ),
      }),
    );
    state.instancePolicyArn = Arn;
  }),
  new ScenarioOutput("createdInstancePolicy", (state) =>
    MESSAGES.createdInstancePolicy
      .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
      .replace("${INSTANCE_POLICY_ARN}", state.instancePolicyArn),
  ),
  new ScenarioOutput(
```

```
    "creatingInstanceRole",
    MESSAGES.creatingInstanceRole.replace(
      "${INSTANCE_ROLE_NAME}",
      NAMES.instanceRoleName,
    ),
  ),
  new ScenarioAction("createInstanceRole", () => {
    const client = new IAMClient({});
    return client.send(
      new CreateRoleCommand({
        RoleName: NAMES.instanceRoleName,
        AssumeRolePolicyDocument: readFileSync(
          join(ROOT, "assume-role-policy.json"),
        ),
      }),
    );
  }),
  new ScenarioOutput(
    "createdInstanceRole",
    MESSAGES.createdInstanceRole.replace(
      "${INSTANCE_ROLE_NAME}",
      NAMES.instanceRoleName,
    ),
  ),
  new ScenarioOutput(
    "attachingPolicyToRole",
    MESSAGES.attachingPolicyToRole
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName)
      .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName),
  ),
  new ScenarioAction("attachPolicyToRole", async (state) => {
    const client = new IAMClient({});
    await client.send(
      new AttachRolePolicyCommand({
        RoleName: NAMES.instanceRoleName,
        PolicyArn: state.instancePolicyArn,
      }),
    );
  }),
  new ScenarioOutput(
    "attachedPolicyToRole",
    MESSAGES.attachedPolicyToRole
      .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
```

```
  ),
  new ScenarioOutput(
    "creatingInstanceProfile",
    MESSAGES.creatingInstanceProfile.replace(
      "${INSTANCE_PROFILE_NAME}",
      NAMES.instanceProfileName,
    ),
  ),
  new ScenarioAction("createInstanceProfile", async (state) => {
    const client = new IAMClient({});
    const {
      InstanceProfile: { Arn },
    } = await client.send(
      new CreateInstanceProfileCommand({
        InstanceProfileName: NAMES.instanceProfileName,
      }),
    );
    state.instanceProfileArn = Arn;

    await waitUntilInstanceProfileExists(
      { client },
      { InstanceProfileName: NAMES.instanceProfileName },
    );
  }),
  new ScenarioOutput("createdInstanceProfile", (state) =>
    MESSAGES.createdInstanceProfile
      .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
      .replace("${INSTANCE_PROFILE_ARN}", state.instanceProfileArn),
  ),
  new ScenarioOutput(
    "addingRoleToInstanceProfile",
    MESSAGES.addingRoleToInstanceProfile
      .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
  ),
  new ScenarioAction("addRoleToInstanceProfile", () => {
    const client = new IAMClient({});
    return client.send(
      new AddRoleToInstanceProfileCommand({
        RoleName: NAMES.instanceRoleName,
        InstanceProfileName: NAMES.instanceProfileName,
      }),
    );
  }),
```

```
  new ScenarioOutput(
    "addedRoleToInstanceProfile",
    MESSAGES.addedRoleToInstanceProfile
      .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
  ),
  ...initParamsSteps,
  new ScenarioOutput("creatingLaunchTemplate", MESSAGES.creatingLaunchTemplate),
  new ScenarioAction("createLaunchTemplate", async () => {
    // snippet-start:[javascript.v3.wkflw.resilient.CreateLaunchTemplate]
    const ssmClient = new SSMClient({});
    const { Parameter } = await ssmClient.send(
      new GetParameterCommand({
        Name: "/aws/service/ami-amazon-linux-latest/amzn2-ami-hvm-x86_64-gp2",
      }),
    );
    const ec2Client = new EC2Client({});
    await ec2Client.send(
      new CreateLaunchTemplateCommand({
        LaunchTemplateName: NAMES.launchTemplateName,
        LaunchTemplateData: {
          InstanceType: "t3.micro",
          ImageId: Parameter.Value,
          IamInstanceProfile: { Name: NAMES.instanceProfileName },
          UserData: readFileSync(
            join(RESOURCES_PATH, "server_startup_script.sh"),
          ).toString("base64"),
          KeyName: NAMES.keyPairName,
        },
      }),
      // snippet-end:[javascript.v3.wkflw.resilient.CreateLaunchTemplate]
    );
  }),
  new ScenarioOutput(
    "createdLaunchTemplate",
    MESSAGES.createdLaunchTemplate.replace(
      "${LAUNCH_TEMPLATE_NAME}",
      NAMES.launchTemplateName,
    ),
  ),
  new ScenarioOutput(
    "creatingAutoScalingGroup",
    MESSAGES.creatingAutoScalingGroup.replace(
      "${AUTO_SCALING_GROUP_NAME}",
```

```
      NAMES.autoScalingGroupName,
    ),
  ),
  new ScenarioAction("createAutoScalingGroup", async (state) => {
    const ec2Client = new EC2Client({});
    const { AvailabilityZones } = await ec2Client.send(
      new DescribeAvailabilityZonesCommand({}),
    );
    state.availabilityZoneNames = AvailabilityZones.map((az) => az.ZoneName);
    const autoScalingClient = new AutoScalingClient({});
    await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
      autoScalingClient.send(
        new CreateAutoScalingGroupCommand({
          AvailabilityZones: state.availabilityZoneNames,
          AutoScalingGroupName: NAMES.autoScalingGroupName,
          LaunchTemplate: {
            LaunchTemplateName: NAMES.launchTemplateName,
            Version: "$Default",
          },
          MinSize: 3,
          MaxSize: 3,
        }),
      ),
    );
  }),
  new ScenarioOutput(
    "createdAutoScalingGroup",
    /**
     * @param {{ availabilityZoneNames: string[] }} state
     */
    (state) =>
      MESSAGES.createdAutoScalingGroup
        .replace("${AUTO_SCALING_GROUP_NAME}", NAMES.autoScalingGroupName)
        .replace(
          "${AVAILABILITY_ZONE_NAMES}",
          state.availabilityZoneNames.join(", "),
        ),
  ),
  new ScenarioInput("confirmContinue", MESSAGES.confirmContinue, {
    type: "confirm",
  }),
  new ScenarioOutput("loadBalancer", MESSAGES.loadBalancer),
  new ScenarioOutput("gettingVpc", MESSAGES.gettingVpc),
  new ScenarioAction("getVpc", async (state) => {
```

```
    // snippet-start:[javascript.v3.wkflw.resilient.DescribeVpcs]
    const client = new EC2Client({});
    const { Vpcs } = await client.send(
      new DescribeVpcsCommand({
        Filters: [{ Name: "is-default", Values: ["true"] }],
      }),
    );
    // snippet-end:[javascript.v3.wkflw.resilient.DescribeVpcs]
    state.defaultVpc = Vpcs[0].VpcId;
  }),
  new ScenarioOutput("gotVpc", (state) =>
    MESSAGES.gotVpc.replace("${VPC_ID}", state.defaultVpc),
  ),
  new ScenarioOutput("gettingSubnets", MESSAGES.gettingSubnets),
  new ScenarioAction("getSubnets", async (state) => {
    // snippet-start:[javascript.v3.wkflw.resilient.DescribeSubnets]
    const client = new EC2Client({});
    const { Subnets } = await client.send(
      new DescribeSubnetsCommand({
        Filters: [
          { Name: "vpc-id", Values: [state.defaultVpc] },
          { Name: "availability-zone", Values: state.availabilityZoneNames },
          { Name: "default-for-az", Values: ["true"] },
        ],
      }),
    );
    // snippet-end:[javascript.v3.wkflw.resilient.DescribeSubnets]
    state.subnets = Subnets.map((subnet) => subnet.SubnetId);
  }),
  new ScenarioOutput(
    "gotSubnets",
    /**
     * @param {{ subnets: string[] }} state
     */
    (state) =>
      MESSAGES.gotSubnets.replace("${SUBNETS}", state.subnets.join(", ")),
  ),
  new ScenarioOutput(
    "creatingLoadBalancerTargetGroup",
    MESSAGES.creatingLoadBalancerTargetGroup.replace(
      "${TARGET_GROUP_NAME}",
      NAMES.loadBalancerTargetGroupName,
    ),
  ),
```

```
  new ScenarioAction("createLoadBalancerTargetGroup", async (state) => {
    // snippet-start:[javascript.v3.wkflw.resilient.CreateTargetGroup]
    const client = new ElasticLoadBalancingV2Client({});
    const { TargetGroups } = await client.send(
      new CreateTargetGroupCommand({
        Name: NAMES.loadBalancerTargetGroupName,
        Protocol: "HTTP",
        Port: 80,
        HealthCheckPath: "/healthcheck",
        HealthCheckIntervalSeconds: 10,
        HealthCheckTimeoutSeconds: 5,
        HealthyThresholdCount: 2,
        UnhealthyThresholdCount: 2,
        VpcId: state.defaultVpc,
      }),
    );
    // snippet-end:[javascript.v3.wkflw.resilient.CreateTargetGroup]
    const targetGroup = TargetGroups[0];
    state.targetGroupArn = targetGroup.TargetGroupArn;
    state.targetGroupProtocol = targetGroup.Protocol;
    state.targetGroupPort = targetGroup.Port;
  }),
  new ScenarioOutput(
    "createdLoadBalancerTargetGroup",
    MESSAGES.createdLoadBalancerTargetGroup.replace(
      "${TARGET_GROUP_NAME}",
      NAMES.loadBalancerTargetGroupName,
    ),
  ),
  new ScenarioOutput(
    "creatingLoadBalancer",
    MESSAGES.creatingLoadBalancer.replace("${LB_NAME}", NAMES.loadBalancerName),
  ),
  new ScenarioAction("createLoadBalancer", async (state) => {
    // snippet-start:[javascript.v3.wkflw.resilient.CreateLoadBalancer]
    const client = new ElasticLoadBalancingV2Client({});
    const { LoadBalancers } = await client.send(
      new CreateLoadBalancerCommand({
        Name: NAMES.loadBalancerName,
        Subnets: state.subnets,
      }),
    );
    state.loadBalancerDns = LoadBalancers[0].DNSName;
    state.loadBalancerArn = LoadBalancers[0].LoadBalancerArn;
```

```
    await waitUntilLoadBalancerAvailable(
      { client },
      { Names: [NAMES.loadBalancerName] },
    );
    // snippet-end:[javascript.v3.wkflw.resilient.CreateLoadBalancer]
  }),
  new ScenarioOutput("createdLoadBalancer", (state) =>
    MESSAGES.createdLoadBalancer
      .replace("${LB_NAME}", NAMES.loadBalancerName)
      .replace("${DNS_NAME}", state.loadBalancerDns),
  ),
  new ScenarioOutput(
    "creatingListener",
    MESSAGES.creatingLoadBalancerListener
      .replace("${LB_NAME}", NAMES.loadBalancerName)
      .replace("${TARGET_GROUP_NAME}", NAMES.loadBalancerTargetGroupName),
  ),
  new ScenarioAction("createListener", async (state) => {
    // snippet-start:[javascript.v3.wkflw.resilient.CreateListener]
    const client = new ElasticLoadBalancingV2Client({});
    const { Listeners } = await client.send(
      new CreateListenerCommand({
        LoadBalancerArn: state.loadBalancerArn,
        Protocol: state.targetGroupProtocol,
        Port: state.targetGroupPort,
        DefaultActions: [
          { Type: "forward", TargetGroupArn: state.targetGroupArn },
        ],
      }),
    );
    // snippet-end:[javascript.v3.wkflw.resilient.CreateListener]
    const listener = Listeners[0];
    state.loadBalancerListenerArn = listener.ListenerArn;
  }),
  new ScenarioOutput("createdListener", (state) =>
    MESSAGES.createdLoadBalancerListener.replace(
      "${LB_LISTENER_ARN}",
      state.loadBalancerListenerArn,
    ),
  ),
  new ScenarioOutput(
    "attachingLoadBalancerTargetGroup",
    MESSAGES.attachingLoadBalancerTargetGroup
      .replace("${TARGET_GROUP_NAME}", NAMES.loadBalancerTargetGroupName)
```

```
      .replace("${AUTO_SCALING_GROUP_NAME}", NAMES.autoScalingGroupName),
  ),
  new ScenarioAction("attachLoadBalancerTargetGroup", async (state) => {
    // snippet-start:[javascript.v3.wkflw.resilient.AttachTargetGroup]
    const client = new AutoScalingClient({});
    await client.send(
      new AttachLoadBalancerTargetGroupsCommand({
        AutoScalingGroupName: NAMES.autoScalingGroupName,
        TargetGroupARNs: [state.targetGroupArn],
      }),
    );
    // snippet-end:[javascript.v3.wkflw.resilient.AttachTargetGroup]
  }),
  new ScenarioOutput(
    "attachedLoadBalancerTargetGroup",
    MESSAGES.attachedLoadBalancerTargetGroup,
  ),
  new ScenarioOutput("verifyingInboundPort", MESSAGES.verifyingInboundPort),
  new ScenarioAction(
    "verifyInboundPort",
    /**
     *
     * @param {{ defaultSecurityGroup: import('@aws-sdk/client-ec2').SecurityGroup}}
 state
     */
    async (state) => {
      const client = new EC2Client({});
      const { SecurityGroups } = await client.send(
        new DescribeSecurityGroupsCommand({
          Filters: [{ Name: "group-name", Values: ["default"] }],
        }),
      );
      if (!SecurityGroups) {
        state.verifyInboundPortError = new Error(MESSAGES.noSecurityGroups);
      }
      state.defaultSecurityGroup = SecurityGroups[0];

      /**
       * @type {string}
       */
      const ipResponse = (await axios.get("http://checkip.amazonaws.com")).data;
      state.myIp = ipResponse.trim();
      const myIpRules = state.defaultSecurityGroup.IpPermissions.filter(
        ({ IpRanges }) =>
```

```
          IpRanges.some(
            ({ CidrIp }) =>
              CidrIp.startsWith(state.myIp) || CidrIp === "0.0.0.0/0",
          ),
      )
        .filter(({ IpProtocol }) => IpProtocol === "tcp")
        .filter(({ FromPort }) => FromPort === 80);

      state.myIpRules = myIpRules;
    },
  ),
  new ScenarioOutput(
    "verifiedInboundPort",
    /**
     * @param {{ myIpRules: any[] }} state
     */
    (state) => {
      if (state.myIpRules.length > 0) {
        return MESSAGES.foundIpRules.replace(
          "${IP_RULES}",
          JSON.stringify(state.myIpRules, null, 2),
        );
      } else {
        return MESSAGES.noIpRules;
      }
    },
  ),
  new ScenarioInput(
    "shouldAddInboundRule",
    /**
     * @param {{ myIpRules: any[] }} state
     */
    (state) => {
      if (state.myIpRules.length > 0) {
        return false;
      } else {
        return MESSAGES.noIpRules;
      }
    },
    { type: "confirm" },
  ),
  new ScenarioAction(
    "addInboundRule",
    /**
```

```
     * @param {{ defaultSecurityGroup: import('@aws-sdk/client-
ec2').SecurityGroup }} state
     */
    async (state) => {
      if (!state.shouldAddInboundRule) {
        return;
      }

      const client = new EC2Client({});
      await client.send(
        new AuthorizeSecurityGroupIngressCommand({
          GroupId: state.defaultSecurityGroup.GroupId,
          CidrIp: `${state.myIp}/32`,
          FromPort: 80,
          ToPort: 80,
          IpProtocol: "tcp",
        }),
      );
    },
  ),
  new ScenarioOutput("addedInboundRule", (state) => {
    if (state.shouldAddInboundRule) {
      return MESSAGES.addedInboundRule.replace("${IP_ADDRESS}", state.myIp);
    } else {
      return false;
    }
  }),
  new ScenarioOutput("verifyingEndpoint", (state) =>
    MESSAGES.verifyingEndpoint.replace("${DNS_NAME}", state.loadBalancerDns),
  ),
  new ScenarioAction("verifyEndpoint", async (state) => {
    try {
      const response = await retry({ intervalInMs: 2000, maxRetries: 30 }, () =>
        axios.get(`http://${state.loadBalancerDns}`),
      );
      state.endpointResponse = JSON.stringify(response.data, null, 2);
    } catch (e) {
      state.verifyEndpointError = e;
    }
  }),
  new ScenarioOutput("verifiedEndpoint", (state) => {
    if (state.verifyEndpointError) {
      console.error(state.verifyEndpointError);
    } else {
```

```
      return MESSAGES.verifiedEndpoint.replace(
        "${ENDPOINT_RESPONSE}",
        state.endpointResponse,
      );
    }
  }),
];
```

创建运行演示的步骤。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { readFileSync } from "node:fs";
import { join } from "node:path";

import axios from "axios";

import {
  DescribeTargetGroupsCommand,
  DescribeTargetHealthCommand,
  ElasticLoadBalancingV2Client,
} from "@aws-sdk/client-elastic-load-balancing-v2";
import {
  DescribeInstanceInformationCommand,
  PutParameterCommand,
  SSMClient,
  SendCommandCommand,
} from "@aws-sdk/client-ssm";
import {
  IAMClient,
  CreatePolicyCommand,
  CreateRoleCommand,
  AttachRolePolicyCommand,
  CreateInstanceProfileCommand,
  AddRoleToInstanceProfileCommand,
  waitUntilInstanceProfileExists,
} from "@aws-sdk/client-iam";
import {
  AutoScalingClient,
  DescribeAutoScalingGroupsCommand,
  TerminateInstanceInAutoScalingGroupCommand,
} from "@aws-sdk/client-auto-scaling";
```

```
import {
  DescribeIamInstanceProfileAssociationsCommand,
  EC2Client,
  RebootInstancesCommand,
  ReplaceIamInstanceProfileAssociationCommand,
} from "@aws-sdk/client-ec2";

import {
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/scenario.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES, RESOURCES_PATH } from "./constants.js";
import { findLoadBalancer } from "./shared.js";

const getRecommendation = new ScenarioAction(
  "getRecommendation",
  async (state) => {
    const loadBalancer = await findLoadBalancer(NAMES.loadBalancerName);
    if (loadBalancer) {
      state.loadBalancerDnsName = loadBalancer.DNSName;
      try {
        state.recommendation = (
          await axios.get(`http://${state.loadBalancerDnsName}`)
        ).data;
      } catch (e) {
        state.recommendation = e instanceof Error ? e.message : e;
      }
    } else {
      throw new Error(MESSAGES.demoFindLoadBalancerError);
    }
  },
);

const getRecommendationResult = new ScenarioOutput(
  "getRecommendationResult",
  (state) =>
    `Recommendation:\n${JSON.stringify(state.recommendation, null, 2)}`,
  { preformatted: true },
);

const getHealthCheck = new ScenarioAction("getHealthCheck", async (state) => {
```

```
  // snippet-start:[javascript.v3.wkflw.resilient.DescribeTargetGroups]
  const client = new ElasticLoadBalancingV2Client({});
  const { TargetGroups } = await client.send(
    new DescribeTargetGroupsCommand({
      Names: [NAMES.loadBalancerTargetGroupName],
    }),
  );
  // snippet-end:[javascript.v3.wkflw.resilient.DescribeTargetGroups]

  // snippet-start:[javascript.v3.wkflw.resilient.DescribeTargetHealth]
  const { TargetHealthDescriptions } = await client.send(
    new DescribeTargetHealthCommand({
      TargetGroupArn: TargetGroups[0].TargetGroupArn,
    }),
  );
  // snippet-end:[javascript.v3.wkflw.resilient.DescribeTargetHealth]
  state.targetHealthDescriptions = TargetHealthDescriptions;
});

const getHealthCheckResult = new ScenarioOutput(
  "getHealthCheckResult",
  /**
   * @param {{ targetHealthDescriptions: import('@aws-sdk/client-elastic-load-
balancing-v2').TargetHealthDescription[]}} state
   */
  (state) => {
    const status = state.targetHealthDescriptions
      .map((th) => `${th.Target.Id}: ${th.TargetHealth.State}`)
      .join("\n");
    return `Health check:\n${status}`;
  },
  { preformatted: true },
);

const loadBalancerLoop = new ScenarioAction(
  "loadBalancerLoop",
  getRecommendation.action,
  {
    whileConfig: {
      inputEquals: true,
      input: new ScenarioInput(
        "loadBalancerCheck",
        MESSAGES.demoLoadBalancerCheck,
        {
```

```
          type: "confirm",
        },
      ),
      output: getRecommendationResult,
    },
  },
);

const healthCheckLoop = new ScenarioAction(
  "healthCheckLoop",
  getHealthCheck.action,
  {
    whileConfig: {
      inputEquals: true,
      input: new ScenarioInput("healthCheck", MESSAGES.demoHealthCheck, {
        type: "confirm",
      }),
      output: getHealthCheckResult,
    },
  },
);

const statusSteps = [
  getRecommendation,
  getRecommendationResult,
  getHealthCheck,
  getHealthCheckResult,
];

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const demoSteps = [
  new ScenarioOutput("header", MESSAGES.demoHeader, { header: true }),
  new ScenarioOutput("sanityCheck", MESSAGES.demoSanityCheck),
  ...statusSteps,
  new ScenarioInput(
    "brokenDependencyConfirmation",
    MESSAGES.demoBrokenDependencyConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("brokenDependency", async (state) => {
    if (!state.brokenDependencyConfirmation) {
      process.exit();
```

```
    } else {
      const client = new SSMClient({});
      state.badTableName = `fake-table-${Date.now()}`;
      await client.send(
        new PutParameterCommand({
          Name: NAMES.ssmTableNameKey,
          Value: state.badTableName,
          Overwrite: true,
          Type: "String",
        }),
      );
    }
  }),
  new ScenarioOutput("testBrokenDependency", (state) =>
    MESSAGES.demoTestBrokenDependency.replace(
      "${TABLE_NAME}",
      state.badTableName,
    ),
  ),
  ...statusSteps,
  new ScenarioInput(
    "staticResponseConfirmation",
    MESSAGES.demoStaticResponseConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("staticResponse", async (state) => {
    if (!state.staticResponseConfirmation) {
      process.exit();
    } else {
      const client = new SSMClient({});
      await client.send(
        new PutParameterCommand({
          Name: NAMES.ssmFailureResponseKey,
          Value: "static",
          Overwrite: true,
          Type: "String",
        }),
      );
    }
  }),
  new ScenarioOutput("testStaticResponse", MESSAGES.demoTestStaticResponse),
  ...statusSteps,
  new ScenarioInput(
    "badCredentialsConfirmation",
```

```
      MESSAGES.demoBadCredentialsConfirmation,
      { type: "confirm" },
    ),
    new ScenarioAction("badCredentialsExit", (state) => {
      if (!state.badCredentialsConfirmation) {
        process.exit();
      }
    }),
    new ScenarioAction("fixDynamoDBName", async () => {
      const client = new SSMClient({});
      await client.send(
        new PutParameterCommand({
          Name: NAMES.ssmTableNameKey,
          Value: NAMES.tableName,
          Overwrite: true,
          Type: "String",
        }),
      );
    }),
    new ScenarioAction(
      "badCredentials",
      /**
       * @param {{ targetInstance: import('@aws-sdk/client-auto-scaling').Instance }}
 state
       */
      async (state) => {
        await createSsmOnlyInstanceProfile();
        const autoScalingClient = new AutoScalingClient({});
        const { AutoScalingGroups } = await autoScalingClient.send(
          new DescribeAutoScalingGroupsCommand({
            AutoScalingGroupNames: [NAMES.autoScalingGroupName],
          }),
        );
        state.targetInstance = AutoScalingGroups[0].Instances[0];
        // snippet-start:
[javascript.v3.wkflw.resilient.DescribeIamInstanceProfileAssociations]
        const ec2Client = new EC2Client({});
        const { IamInstanceProfileAssociations } = await ec2Client.send(
          new DescribeIamInstanceProfileAssociationsCommand({
            Filters: [
              { Name: "instance-id", Values: [state.targetInstance.InstanceId] },
            ],
          }),
        );
```

```
      // snippet-end:
[javascript.v3.wkflw.resilient.DescribeIamInstanceProfileAssociations]
      state.instanceProfileAssociationId =
        IamInstanceProfileAssociations[0].AssociationId;
      // snippet-start:
[javascript.v3.wkflw.resilient.ReplaceIamInstanceProfileAssociation]
      await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
        ec2Client.send(
          new ReplaceIamInstanceProfileAssociationCommand({
            AssociationId: state.instanceProfileAssociationId,
            IamInstanceProfile: { Name: NAMES.ssmOnlyInstanceProfileName },
          }),
        ),
      );
      // snippet-end:
[javascript.v3.wkflw.resilient.ReplaceIamInstanceProfileAssociation]

      await ec2Client.send(
        new RebootInstancesCommand({
          InstanceIds: [state.targetInstance.InstanceId],
        }),
      );

      const ssmClient = new SSMClient({});
      await retry({ intervalInMs: 20000, maxRetries: 15 }, async () => {
        const { InstanceInformationList } = await ssmClient.send(
          new DescribeInstanceInformationCommand({}),
        );

        const instance = InstanceInformationList.find(
          (info) => info.InstanceId === state.targetInstance.InstanceId,
        );

        if (!instance) {
          throw new Error("Instance not found.");
        }
      });

      await ssmClient.send(
        new SendCommandCommand({
          InstanceIds: [state.targetInstance.InstanceId],
          DocumentName: "AWS-RunShellScript",
          Parameters: { commands: ["cd / && sudo python3 server.py 80"] },
        }),
```

```
      );
    },
  ),
  new ScenarioOutput(
    "testBadCredentials",
    /**
     * @param {{ targetInstance: import('@aws-sdk/client-ssm').InstanceInformation}}
 state
     */
    (state) =>
      MESSAGES.demoTestBadCredentials.replace(
        "${INSTANCE_ID}",
        state.targetInstance.InstanceId,
      ),
  ),
  loadBalancerLoop,
  new ScenarioInput(
    "deepHealthCheckConfirmation",
    MESSAGES.demoDeepHealthCheckConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("deepHealthCheckExit", (state) => {
    if (!state.deepHealthCheckConfirmation) {
      process.exit();
    }
  }),
  new ScenarioAction("deepHealthCheck", async () => {
    const client = new SSMClient({});
    await client.send(
      new PutParameterCommand({
        Name: NAMES.ssmHealthCheckKey,
        Value: "deep",
        Overwrite: true,
        Type: "String",
      }),
    );
  }),
  new ScenarioOutput("testDeepHealthCheck", MESSAGES.demoTestDeepHealthCheck),
  healthCheckLoop,
  loadBalancerLoop,
  new ScenarioInput(
    "killInstanceConfirmation",
    /**
```

```
     * @param {{ targetInstance: import('@aws-sdk/client-
ssm').InstanceInformation }} state
     */
    (state) =>
      MESSAGES.demoKillInstanceConfirmation.replace(
        "${INSTANCE_ID}",
        state.targetInstance.InstanceId,
      ),
    { type: "confirm" },
  ),
  new ScenarioAction("killInstanceExit", (state) => {
    if (!state.killInstanceConfirmation) {
      process.exit();
    }
  }),
  new ScenarioAction(
    "killInstance",
    /**
     * @param {{ targetInstance: import('@aws-sdk/client-
ssm').InstanceInformation }} state
     */
    async (state) => {
      const client = new AutoScalingClient({});
      await client.send(
        new TerminateInstanceInAutoScalingGroupCommand({
          InstanceId: state.targetInstance.InstanceId,
          ShouldDecrementDesiredCapacity: false,
        }),
      );
    },
  ),
  new ScenarioOutput("testKillInstance", MESSAGES.demoTestKillInstance),
  healthCheckLoop,
  loadBalancerLoop,
  new ScenarioInput("failOpenConfirmation", MESSAGES.demoFailOpenConfirmation, {
    type: "confirm",
  }),
  new ScenarioAction("failOpenExit", (state) => {
    if (!state.failOpenConfirmation) {
      process.exit();
    }
  }),
  new ScenarioAction("failOpen", () => {
    const client = new SSMClient({});
```

```
    return client.send(
      new PutParameterCommand({
        Name: NAMES.ssmTableNameKey,
        Value: `fake-table-${Date.now()}`,
        Overwrite: true,
        Type: "String",
      }),
    );
  }),
  new ScenarioOutput("testFailOpen", MESSAGES.demoFailOpenTest),
  healthCheckLoop,
  loadBalancerLoop,
  new ScenarioInput(
    "resetTableConfirmation",
    MESSAGES.demoResetTableConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("resetTableExit", (state) => {
    if (!state.resetTableConfirmation) {
      process.exit();
    }
  }),
  new ScenarioAction("resetTable", async () => {
    const client = new SSMClient({});
    await client.send(
      new PutParameterCommand({
        Name: NAMES.ssmTableNameKey,
        Value: NAMES.tableName,
        Overwrite: true,
        Type: "String",
      }),
    );
  }),
  new ScenarioOutput("testResetTable", MESSAGES.demoTestResetTable),
  healthCheckLoop,
  loadBalancerLoop,
];

async function createSsmOnlyInstanceProfile() {
  const iamClient = new IAMClient({});
  const { Policy } = await iamClient.send(
    new CreatePolicyCommand({
      PolicyName: NAMES.ssmOnlyPolicyName,
      PolicyDocument: readFileSync(
```

```
        join(RESOURCES_PATH, "ssm_only_policy.json"),
      ),
    }),
  );
  await iamClient.send(
    new CreateRoleCommand({
      RoleName: NAMES.ssmOnlyRoleName,
      AssumeRolePolicyDocument: JSON.stringify({
        Version: "2012-10-17",
        Statement: [
          {
            Effect: "Allow",
            Principal: { Service: "ec2.amazonaws.com" },
            Action: "sts:AssumeRole",
          },
        ],
      }),
    }),
  );
  await iamClient.send(
    new AttachRolePolicyCommand({
      RoleName: NAMES.ssmOnlyRoleName,
      PolicyArn: Policy.Arn,
    }),
  );
  await iamClient.send(
    new AttachRolePolicyCommand({
      RoleName: NAMES.ssmOnlyRoleName,
      PolicyArn: "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore",
    }),
  );
  // snippet-start:[javascript.v3.wkflw.resilient.CreateInstanceProfile]
  const { InstanceProfile } = await iamClient.send(
    new CreateInstanceProfileCommand({
      InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
    }),
  );
  await waitUntilInstanceProfileExists(
    { client: iamClient },
    { InstanceProfileName: NAMES.ssmOnlyInstanceProfileName },
  );
  // snippet-end:[javascript.v3.wkflw.resilient.CreateInstanceProfile]
  await iamClient.send(
    new AddRoleToInstanceProfileCommand({
```

```
      InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
      RoleName: NAMES.ssmOnlyRoleName,
    }),
  );

  return InstanceProfile;
}
```

创建销毁所有资源的步骤。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { unlinkSync } from "node:fs";

import { DynamoDBClient, DeleteTableCommand } from "@aws-sdk/client-dynamodb";
import {
  EC2Client,
  DeleteKeyPairCommand,
  DeleteLaunchTemplateCommand,
} from "@aws-sdk/client-ec2";
import {
  IAMClient,
  DeleteInstanceProfileCommand,
  RemoveRoleFromInstanceProfileCommand,
  DeletePolicyCommand,
  DeleteRoleCommand,
  DetachRolePolicyCommand,
  paginateListPolicies,
} from "@aws-sdk/client-iam";
import {
  AutoScalingClient,
  DeleteAutoScalingGroupCommand,
  TerminateInstanceInAutoScalingGroupCommand,
  UpdateAutoScalingGroupCommand,
  paginateDescribeAutoScalingGroups,
} from "@aws-sdk/client-auto-scaling";
import {
  DeleteLoadBalancerCommand,
  DeleteTargetGroupCommand,
  DescribeTargetGroupsCommand,
  ElasticLoadBalancingV2Client,
} from "@aws-sdk/client-elastic-load-balancing-v2";
```

```
import {
  ScenarioOutput,
  ScenarioInput,
  ScenarioAction,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES } from "./constants.js";
import { findLoadBalancer } from "./shared.js";

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const destroySteps = [
  new ScenarioInput("destroy", MESSAGES.destroy, { type: "confirm" }),
  new ScenarioAction(
    "abort",
    (state) => state.destroy === false && process.exit(),
  ),
  new ScenarioAction("deleteTable", async (c) => {
    try {
      const client = new DynamoDBClient({});
      await client.send(new DeleteTableCommand({ TableName: NAMES.tableName }));
    } catch (e) {
      c.deleteTableError = e;
    }
  }),
  new ScenarioOutput("deleteTableResult", (state) => {
    if (state.deleteTableError) {
      console.error(state.deleteTableError);
      return MESSAGES.deleteTableError.replace(
        "${TABLE_NAME}",
        NAMES.tableName,
      );
    } else {
      return MESSAGES.deletedTable.replace("${TABLE_NAME}", NAMES.tableName);
    }
  }),
  new ScenarioAction("deleteKeyPair", async (state) => {
    try {
      const client = new EC2Client({});
      await client.send(
        new DeleteKeyPairCommand({ KeyName: NAMES.keyPairName }),
```

```
      );
      unlinkSync(`${NAMES.keyPairName}.pem`);
    } catch (e) {
      state.deleteKeyPairError = e;
    }
  }),
  new ScenarioOutput("deleteKeyPairResult", (state) => {
    if (state.deleteKeyPairError) {
      console.error(state.deleteKeyPairError);
      return MESSAGES.deleteKeyPairError.replace(
        "${KEY_PAIR_NAME}",
        NAMES.keyPairName,
      );
    } else {
      return MESSAGES.deletedKeyPair.replace(
        "${KEY_PAIR_NAME}",
        NAMES.keyPairName,
      );
    }
  }),
  new ScenarioAction("detachPolicyFromRole", async (state) => {
    try {
      const client = new IAMClient({});
      const policy = await findPolicy(NAMES.instancePolicyName);

      if (!policy) {
        state.detachPolicyFromRoleError = new Error(
          `Policy ${NAMES.instancePolicyName} not found.`,
        );
      } else {
        await client.send(
          new DetachRolePolicyCommand({
            RoleName: NAMES.instanceRoleName,
            PolicyArn: policy.Arn,
          }),
        );
      }
    } catch (e) {
      state.detachPolicyFromRoleError = e;
    }
  }),
  new ScenarioOutput("detachedPolicyFromRole", (state) => {
    if (state.detachPolicyFromRoleError) {
      console.error(state.detachPolicyFromRoleError);
```

```
      return MESSAGES.detachPolicyFromRoleError
        .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    } else {
      return MESSAGES.detachedPolicyFromRole
        .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    }
  }),
  new ScenarioAction("deleteInstancePolicy", async (state) => {
    const client = new IAMClient({});
    const policy = await findPolicy(NAMES.instancePolicyName);

    if (!policy) {
      state.deletePolicyError = new Error(
        `Policy ${NAMES.instancePolicyName} not found.`,
      );
    } else {
      return client.send(
        new DeletePolicyCommand({
          PolicyArn: policy.Arn,
        }),
      );
    }
  }),
  new ScenarioOutput("deletePolicyResult", (state) => {
    if (state.deletePolicyError) {
      console.error(state.deletePolicyError);
      return MESSAGES.deletePolicyError.replace(
        "${INSTANCE_POLICY_NAME}",
        NAMES.instancePolicyName,
      );
    } else {
      return MESSAGES.deletedPolicy.replace(
        "${INSTANCE_POLICY_NAME}",
        NAMES.instancePolicyName,
      );
    }
  }),
  new ScenarioAction("removeRoleFromInstanceProfile", async (state) => {
    try {
      const client = new IAMClient({});
      await client.send(
        new RemoveRoleFromInstanceProfileCommand({
```

```
          RoleName: NAMES.instanceRoleName,
          InstanceProfileName: NAMES.instanceProfileName,
        }),
      );
    } catch (e) {
      state.removeRoleFromInstanceProfileError = e;
    }
  }),
  new ScenarioOutput("removeRoleFromInstanceProfileResult", (state) => {
    if (state.removeRoleFromInstanceProfile) {
      console.error(state.removeRoleFromInstanceProfileError);
      return MESSAGES.removeRoleFromInstanceProfileError
        .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    } else {
      return MESSAGES.removedRoleFromInstanceProfile
        .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    }
  }),
  new ScenarioAction("deleteInstanceRole", async (state) => {
    try {
      const client = new IAMClient({});
      await client.send(
        new DeleteRoleCommand({
          RoleName: NAMES.instanceRoleName,
        }),
      );
    } catch (e) {
      state.deleteInstanceRoleError = e;
    }
  }),
  new ScenarioOutput("deleteInstanceRoleResult", (state) => {
    if (state.deleteInstanceRoleError) {
      console.error(state.deleteInstanceRoleError);
      return MESSAGES.deleteInstanceRoleError.replace(
        "${INSTANCE_ROLE_NAME}",
        NAMES.instanceRoleName,
      );
    } else {
      return MESSAGES.deletedInstanceRole.replace(
        "${INSTANCE_ROLE_NAME}",
        NAMES.instanceRoleName,
      );
```

```
    }
  }),
  new ScenarioAction("deleteInstanceProfile", async (state) => {
    try {
      // snippet-start:[javascript.v3.wkflw.resilient.DeleteInstanceProfile]
      const client = new IAMClient({});
      await client.send(
        new DeleteInstanceProfileCommand({
          InstanceProfileName: NAMES.instanceProfileName,
        }),
      );
      // snippet-end:[javascript.v3.wkflw.resilient.DeleteInstanceProfile]
    } catch (e) {
      state.deleteInstanceProfileError = e;
    }
  }),
  new ScenarioOutput("deleteInstanceProfileResult", (state) => {
    if (state.deleteInstanceProfileError) {
      console.error(state.deleteInstanceProfileError);
      return MESSAGES.deleteInstanceProfileError.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.instanceProfileName,
      );
    } else {
      return MESSAGES.deletedInstanceProfile.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.instanceProfileName,
      );
    }
  }),
  new ScenarioAction("deleteAutoScalingGroup", async (state) => {
    try {
      await terminateGroupInstances(NAMES.autoScalingGroupName);
      await retry({ intervalInMs: 60000, maxRetries: 60 }, async () => {
        await deleteAutoScalingGroup(NAMES.autoScalingGroupName);
      });
    } catch (e) {
      state.deleteAutoScalingGroupError = e;
    }
  }),
  new ScenarioOutput("deleteAutoScalingGroupResult", (state) => {
    if (state.deleteAutoScalingGroupError) {
      console.error(state.deleteAutoScalingGroupError);
      return MESSAGES.deleteAutoScalingGroupError.replace(
```

```
          "${AUTO_SCALING_GROUP_NAME}",
          NAMES.autoScalingGroupName,
        );
      } else {
        return MESSAGES.deletedAutoScalingGroup.replace(
          "${AUTO_SCALING_GROUP_NAME}",
          NAMES.autoScalingGroupName,
        );
      }
    }),
    new ScenarioAction("deleteLaunchTemplate", async (state) => {
      const client = new EC2Client({});
      try {
        // snippet-start:[javascript.v3.wkflw.resilient.DeleteLaunchTemplate]
        await client.send(
          new DeleteLaunchTemplateCommand({
            LaunchTemplateName: NAMES.launchTemplateName,
          }),
        );
        // snippet-end:[javascript.v3.wkflw.resilient.DeleteLaunchTemplate]
      } catch (e) {
        state.deleteLaunchTemplateError = e;
      }
    }),
    new ScenarioOutput("deleteLaunchTemplateResult", (state) => {
      if (state.deleteLaunchTemplateError) {
        console.error(state.deleteLaunchTemplateError);
        return MESSAGES.deleteLaunchTemplateError.replace(
          "${LAUNCH_TEMPLATE_NAME}",
          NAMES.launchTemplateName,
        );
      } else {
        return MESSAGES.deletedLaunchTemplate.replace(
          "${LAUNCH_TEMPLATE_NAME}",
          NAMES.launchTemplateName,
        );
      }
    }),
    new ScenarioAction("deleteLoadBalancer", async (state) => {
      try {
        // snippet-start:[javascript.v3.wkflw.resilient.DeleteLoadBalancer]
        const client = new ElasticLoadBalancingV2Client({});
        const loadBalancer = await findLoadBalancer(NAMES.loadBalancerName);
        await client.send(
```

```
        new DeleteLoadBalancerCommand({
          LoadBalancerArn: loadBalancer.LoadBalancerArn,
        }),
      );
      await retry({ intervalInMs: 1000, maxRetries: 60 }, async () => {
        const lb = await findLoadBalancer(NAMES.loadBalancerName);
        if (lb) {
          throw new Error("Load balancer still exists.");
        }
      });
      // snippet-end:[javascript.v3.wkflw.resilient.DeleteLoadBalancer]
    } catch (e) {
      state.deleteLoadBalancerError = e;
    }
  }),
  new ScenarioOutput("deleteLoadBalancerResult", (state) => {
    if (state.deleteLoadBalancerError) {
      console.error(state.deleteLoadBalancerError);
      return MESSAGES.deleteLoadBalancerError.replace(
        "${LB_NAME}",
        NAMES.loadBalancerName,
      );
    } else {
      return MESSAGES.deletedLoadBalancer.replace(
        "${LB_NAME}",
        NAMES.loadBalancerName,
      );
    }
  }),
  new ScenarioAction("deleteLoadBalancerTargetGroup", async (state) => {
    // snippet-start:[javascript.v3.wkflw.resilient.DeleteTargetGroup]
    const client = new ElasticLoadBalancingV2Client({});
    try {
      const { TargetGroups } = await client.send(
        new DescribeTargetGroupsCommand({
          Names: [NAMES.loadBalancerTargetGroupName],
        }),
      );

      await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
        client.send(
          new DeleteTargetGroupCommand({
            TargetGroupArn: TargetGroups[0].TargetGroupArn,
          }),
```

```
        ),
      );
    } catch (e) {
      state.deleteLoadBalancerTargetGroupError = e;
    }
    // snippet-end:[javascript.v3.wkflw.resilient.DeleteTargetGroup]
  }),
  new ScenarioOutput("deleteLoadBalancerTargetGroupResult", (state) => {
    if (state.deleteLoadBalancerTargetGroupError) {
      console.error(state.deleteLoadBalancerTargetGroupError);
      return MESSAGES.deleteLoadBalancerTargetGroupError.replace(
        "${TARGET_GROUP_NAME}",
        NAMES.loadBalancerTargetGroupName,
      );
    } else {
      return MESSAGES.deletedLoadBalancerTargetGroup.replace(
        "${TARGET_GROUP_NAME}",
        NAMES.loadBalancerTargetGroupName,
      );
    }
  }),
  new ScenarioAction("detachSsmOnlyRoleFromProfile", async (state) => {
    try {
      const client = new IAMClient({});
      await client.send(
        new RemoveRoleFromInstanceProfileCommand({
          InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
          RoleName: NAMES.ssmOnlyRoleName,
        }),
      );
    } catch (e) {
      state.detachSsmOnlyRoleFromProfileError = e;
    }
  }),
  new ScenarioOutput("detachSsmOnlyRoleFromProfileResult", (state) => {
    if (state.detachSsmOnlyRoleFromProfileError) {
      console.error(state.detachSsmOnlyRoleFromProfileError);
      return MESSAGES.detachSsmOnlyRoleFromProfileError
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${PROFILE_NAME}", NAMES.ssmOnlyInstanceProfileName);
    } else {
      return MESSAGES.detachedSsmOnlyRoleFromProfile
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${PROFILE_NAME}", NAMES.ssmOnlyInstanceProfileName);
```

```
    }
  }),
  new ScenarioAction("detachSsmOnlyCustomRolePolicy", async (state) => {
    try {
      const iamClient = new IAMClient({});
      const ssmOnlyPolicy = await findPolicy(NAMES.ssmOnlyPolicyName);
      await iamClient.send(
        new DetachRolePolicyCommand({
          RoleName: NAMES.ssmOnlyRoleName,
          PolicyArn: ssmOnlyPolicy.Arn,
        }),
      );
    } catch (e) {
      state.detachSsmOnlyCustomRolePolicyError = e;
    }
  }),
  new ScenarioOutput("detachSsmOnlyCustomRolePolicyResult", (state) => {
    if (state.detachSsmOnlyCustomRolePolicyError) {
      console.error(state.detachSsmOnlyCustomRolePolicyError);
      return MESSAGES.detachSsmOnlyCustomRolePolicyError
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${POLICY_NAME}", NAMES.ssmOnlyPolicyName);
    } else {
      return MESSAGES.detachedSsmOnlyCustomRolePolicy
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${POLICY_NAME}", NAMES.ssmOnlyPolicyName);
    }
  }),
  new ScenarioAction("detachSsmOnlyAWSRolePolicy", async (state) => {
    try {
      const iamClient = new IAMClient({});
      await iamClient.send(
        new DetachRolePolicyCommand({
          RoleName: NAMES.ssmOnlyRoleName,
          PolicyArn: "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore",
        }),
      );
    } catch (e) {
      state.detachSsmOnlyAWSRolePolicyError = e;
    }
  }),
  new ScenarioOutput("detachSsmOnlyAWSRolePolicyResult", (state) => {
    if (state.detachSsmOnlyAWSRolePolicyError) {
      console.error(state.detachSsmOnlyAWSRolePolicyError);
```

```
      return MESSAGES.detachSsmOnlyAWSRolePolicyError
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${POLICY_NAME}", "AmazonSSMManagedInstanceCore");
    } else {
      return MESSAGES.detachedSsmOnlyAWSRolePolicy
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${POLICY_NAME}", "AmazonSSMManagedInstanceCore");
    }
  }),
  new ScenarioAction("deleteSsmOnlyInstanceProfile", async (state) => {
    try {
      const iamClient = new IAMClient({});
      await iamClient.send(
        new DeleteInstanceProfileCommand({
          InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
        }),
      );
    } catch (e) {
      state.deleteSsmOnlyInstanceProfileError = e;
    }
  }),
  new ScenarioOutput("deleteSsmOnlyInstanceProfileResult", (state) => {
    if (state.deleteSsmOnlyInstanceProfileError) {
      console.error(state.deleteSsmOnlyInstanceProfileError);
      return MESSAGES.deleteSsmOnlyInstanceProfileError.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.ssmOnlyInstanceProfileName,
      );
    } else {
      return MESSAGES.deletedSsmOnlyInstanceProfile.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.ssmOnlyInstanceProfileName,
      );
    }
  }),
  new ScenarioAction("deleteSsmOnlyPolicy", async (state) => {
    try {
      const iamClient = new IAMClient({});
      const ssmOnlyPolicy = await findPolicy(NAMES.ssmOnlyPolicyName);
      await iamClient.send(
        new DeletePolicyCommand({
          PolicyArn: ssmOnlyPolicy.Arn,
        }),
      );
```

```
    } catch (e) {
      state.deleteSsmOnlyPolicyError = e;
    }
  }),
  new ScenarioOutput("deleteSsmOnlyPolicyResult", (state) => {
    if (state.deleteSsmOnlyPolicyError) {
      console.error(state.deleteSsmOnlyPolicyError);
      return MESSAGES.deleteSsmOnlyPolicyError.replace(
        "${POLICY_NAME}",
        NAMES.ssmOnlyPolicyName,
      );
    } else {
      return MESSAGES.deletedSsmOnlyPolicy.replace(
        "${POLICY_NAME}",
        NAMES.ssmOnlyPolicyName,
      );
    }
  }),
  new ScenarioAction("deleteSsmOnlyRole", async (state) => {
    try {
      const iamClient = new IAMClient({});
      await iamClient.send(
        new DeleteRoleCommand({
          RoleName: NAMES.ssmOnlyRoleName,
        }),
      );
    } catch (e) {
      state.deleteSsmOnlyRoleError = e;
    }
  }),
  new ScenarioOutput("deleteSsmOnlyRoleResult", (state) => {
    if (state.deleteSsmOnlyRoleError) {
      console.error(state.deleteSsmOnlyRoleError);
      return MESSAGES.deleteSsmOnlyRoleError.replace(
        "${ROLE_NAME}",
        NAMES.ssmOnlyRoleName,
      );
    } else {
      return MESSAGES.deletedSsmOnlyRole.replace(
        "${ROLE_NAME}",
        NAMES.ssmOnlyRoleName,
      );
    }
  }),
```

```
];

/**
 * @param {string} policyName
 */
async function findPolicy(policyName) {
  const client = new IAMClient({});
  const paginatedPolicies = paginateListPolicies({ client }, {});
  for await (const page of paginatedPolicies) {
    const policy = page.Policies.find((p) => p.PolicyName === policyName);
    if (policy) {
      return policy;
    }
  }
}

/**
 * @param {string} groupName
 */
async function deleteAutoScalingGroup(groupName) {
  const client = new AutoScalingClient({});
  try {
    await client.send(
      new DeleteAutoScalingGroupCommand({
        AutoScalingGroupName: groupName,
      }),
    );
  } catch (err) {
    if (!(err instanceof Error)) {
      throw err;
    } else {
      console.log(err.name);
      throw err;
    }
  }
}

/**
 * @param {string} groupName
 */
async function terminateGroupInstances(groupName) {
  const autoScalingClient = new AutoScalingClient({});
  const group = await findAutoScalingGroup(groupName);
  await autoScalingClient.send(
```

```
    new UpdateAutoScalingGroupCommand({
      AutoScalingGroupName: group.AutoScalingGroupName,
      MinSize: 0,
    }),
  );
  for (const i of group.Instances) {
    await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
      autoScalingClient.send(
        new TerminateInstanceInAutoScalingGroupCommand({
          InstanceId: i.InstanceId,
          ShouldDecrementDesiredCapacity: true,
        }),
      ),
    );
  }
}

async function findAutoScalingGroup(groupName) {
  const client = new AutoScalingClient({});
  const paginatedGroups = paginateDescribeAutoScalingGroups({ client }, {});
  for await (const page of paginatedGroups) {
    const group = page.AutoScalingGroups.find(
      (g) => g.AutoScalingGroupName === groupName,
    );
    if (group) {
      return group;
    }
  }
  throw new Error(`Auto scaling group ${groupName} not found.`);
}
```

- 有关 API 详细信息，请参阅《Amazon SDK for JavaScript API 参考》中的以下主题。
  - AttachLoadBalancerTargetGroups
  - CreateAutoScalingGroup
  - CreateInstanceProfile
  - CreateLaunchTemplate
  - CreateListener
  - CreateLoadBalancer
  - CreateTargetGroup

- DeleteAutoScalingGroup
- DeleteInstanceProfile
- DeleteLaunchTemplate
- DeleteLoadBalancer
- DeleteTargetGroup
- DescribeAutoScalingGroups
- DescribeAvailabilityZones
- DescribeIamInstanceProfileAssociations
- DescribeInstances
- DescribeLoadBalancers
- DescribeSubnets
- DescribeTargetGroups
- DescribeTargetHealth
- DescribeVpcs
- RebootInstances
- ReplaceIamInstanceProfileAssociation
- TerminateInstanceInAutoScalingGroup
- UpdateAutoScalingGroup

使用适用于 JavaScript (v3) 的 SDK 的 Amazon Bedrock 示例

以下代码示例向您展示了如何使用带有 Amazon Bedrock 的 Amazon SDK for JavaScript (v3) 来执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景 是展示如何通过在同一服务中调用多个函数来完成特定任务的代码示例。

每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关如何在上下文中设置和运行代码的说明。

开始使用

开始使用 Amazon Bedrock

以下代码示例演示了如何开始使用 Amazon Bedrock。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

import { fileURLToPath } from "url";

import {
  BedrockClient,
  ListFoundationModelsCommand,
} from "@aws-sdk/client-bedrock";

const REGION = "us-east-1";
const client = new BedrockClient({ region: REGION });

export const main = async () => {
  const command = new ListFoundationModelsCommand({});

  const response = await client.send(command);
  const models = response.modelSummaries;

  console.log("Listing the available Bedrock foundation models:");

  for (let model of models) {
    console.log("=".repeat(42));
    console.log(` Model: ${model.modelId}`);
    console.log("-".repeat(42));
    console.log(` Name: ${model.modelName}`);
    console.log(` Provider: ${model.providerName}`);
    console.log(` Model ARN: ${model.modelArn}`);
    console.log(` Input modalities: ${model.inputModalities}`);
    console.log(` Output modalities: ${model.outputModalities}`);
    console.log(` Supported customizations: ${model.customizationsSupported}`);
```

```
    console.log(` Supported inference types: ${model.inferenceTypesSupported}`);
    console.log(` Lifecycle status: ${model.modelLifecycle.status}`);
    console.log("=".repeat(42) + "\n");
  }

  const active = models.filter(
    (m) => m.modelLifecycle.status === "ACTIVE",
  ).length;
  const legacy = models.filter(
    (m) => m.modelLifecycle.status === "LEGACY",
  ).length;

  console.log(
    `There are ${active} active and ${legacy} legacy foundation models in
 ${REGION}.`,
  );

  return response;
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  await main();
}
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考ListFoundationModels中的。

主题

- 操作

操作

获取有关 Amazon Bedrock 基础模型的详细信息

以下代码示例演示了如何获取有关 Amazon Bedrock 基础模型的详细信息。

适用于 JavaScript (v3) 的软件开发工具包

Note

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

获取有关基础模型的详细信息。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

import { fileURLToPath } from "url";

import {
  BedrockClient,
  GetFoundationModelCommand,
} from "@aws-sdk/client-bedrock";

/**
 * Get details about an Amazon Bedrock foundation model.
 *
 * @return {FoundationModelDetails} - The list of available bedrock foundation
 models.
 */
export const getFoundationModel = async () => {
  const client = new BedrockClient();

  const command = new GetFoundationModelCommand({
    modelIdentifier: "amazon.titan-embed-text-v1",
  });

  const response = await client.send(command);

  return response.modelDetails;
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const model = await getFoundationModel();
  console.log(model);
}
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考GetFoundationModel中的。

列出可用的 Amazon Bedrock 基础模型

以下代码示例演示了如何列出可用的 Amazon Bedrock 基础模型。

适用于 JavaScript (v3) 的软件开发工具包

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

列出可用的基础模型。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

import { fileURLToPath } from "url";

import {
  BedrockClient,
  ListFoundationModelsCommand,
} from "@aws-sdk/client-bedrock";

/**
 * List the available Amazon Bedrock foundation models.
 *
 * @return {FoundationModelSummary[]} - The list of available bedrock foundation
 models.
 */
export const listFoundationModels = async () => {
  const client = new BedrockClient();

  const input = {
    // byProvider: 'STRING_VALUE',
    // byCustomizationType: 'FINE_TUNING' || 'CONTINUED_PRE_TRAINING',
    // byOutputModality: 'TEXT' || 'IMAGE' || 'EMBEDDING',
    // byInferenceType: 'ON_DEMAND' || 'PROVISIONED',
```

```
  };

  const command = new ListFoundationModelsCommand(input);

  const response = await client.send(command);

  return response.modelSummaries;
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const models = await listFoundationModels();
  console.log(models);
}
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考ListFoundationModels中的。

使用适用于 JavaScript (v3) 的 SDK 的亚马逊 Bedrock 运行时示例

以下代码示例向您展示了如何使用带有 Amazon Bedrock Runtime 的 Amazon SDK for JavaScript (v3) 来执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景 是展示如何通过在同一服务中调用多个函数来完成特定任务的代码示例。

每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关如何在上下文中设置和运行代码的说明。

主题

- 操作

操作

使用 AI21 Labs Jurassic-2 生成文本

以下代码示例演示了如何通过在 Amazon Bedrock 上调用 AI21 Labs Jurassic-2 模型来生成文本。

适用于 JavaScript (v3) 的软件开发工具包

Note

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

调用 AI21 Labs Jurassic-2 基础模型以生成文本。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

import { fileURLToPath } from "url";

import {
  AccessDeniedException,
  BedrockRuntimeClient,
  InvokeModelCommand,
} from "@aws-sdk/client-bedrock-runtime";

/**
 * @typedef {Object} Data
 * @property {string} text
 *
 * @typedef {Object} Completion
 * @property {Data} data
 *
 * @typedef {Object} ResponseBody
 * @property {Completion[]} completions
 */

/**
 * Invokes the AI21 Labs Jurassic-2 large-language model to run an inference
 * using the input provided in the request body.
 *
 * @param {string} prompt - The prompt that you want Jurassic-2 to complete.
 * @returns {string} The inference response (completion) from the model.
 */
export const invokeJurassic2 = async (prompt) => {
  const client = new BedrockRuntimeClient({ region: "us-east-1" });

  const modelId = "ai21.j2-mid-v1";
```

```
  /* The different model providers have individual request and response formats.
   * For the format, ranges, and default values for AI21 Labs Jurassic-2, refer to:
   * https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters-
jurassic2.html
   */
  const payload = {
    prompt,
    maxTokens: 500,
    temperature: 0.5,
  };

  const command = new InvokeModelCommand({
    body: JSON.stringify(payload),
    contentType: "application/json",
    accept: "application/json",
    modelId,
  });

  try {
    const response = await client.send(command);
    const decodedResponseBody = new TextDecoder().decode(response.body);

    /** @type {ResponseBody} */
    const responseBody = JSON.parse(decodedResponseBody);

    return responseBody.completions[0].data.text;
  } catch (err) {
    if (err instanceof AccessDeniedException) {
      console.error(
        `Access denied. Ensure you have the correct permissions to invoke
 ${modelId}.`,
      );
    } else {
      throw err;
    }
  }
};

// Invoke the function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const prompt = 'Complete the following: "Once upon a time..."';
  console.log("\nModel: AI21 Labs Jurassic-2");
  console.log(`Prompt: ${prompt}`);
```

```
  const completion = await invokeJurassic2(prompt);
  console.log("Completion:");
  console.log(completion);
  console.log("\n");
}
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考InvokeModel中的。

使用 Amazon Titan Text G1 生成文本

以下代码示例显示了如何在 Amazon Bedrock 上调用 Amazon Titan Text G1 模型来生成文本。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

调用 Amazon Titan Text G1 基础模型来生成文本。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

import { fileURLToPath } from "url";

import {
  AccessDeniedException,
  BedrockRuntimeClient,
  InvokeModelCommand,
} from "@aws-sdk/client-bedrock-runtime";

/**
 * @typedef {Object} ResponseBody
 * @property {Object[]} results
 */

/**
 * Invokes the Titan Text G1 - Express model to run an inference
```

```
 * using the input provided in the request body.
 *
 * @param {string} prompt - The prompt that you want Titan Text Express to complete.
 * @returns {object[]} The inference response (results) from the model.
 */
export const invokeTitanTextExpressV1 = async (prompt) => {
  const client = new BedrockRuntimeClient({ region: "us-east-1" });

  const modelId = "amazon.titan-text-express-v1";

  /* The different model providers have individual request and response formats.
   * For the format, ranges, and default values for Titan text, refer to:
   * https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters-titan-
text.html
   */
  const textGenerationConfig = {
    maxTokenCount: 4096,
    stopSequences: [],
    temperature: 0,
    topP: 1,
  };

  const payload = {
    inputText: prompt,
    textGenerationConfig,
  };

  const command = new InvokeModelCommand({
    body: JSON.stringify(payload),
    contentType: "application/json",
    accept: "application/json",
    modelId,
  });

  try {
    const response = await client.send(command);
    const decodedResponseBody = new TextDecoder().decode(response.body);

    /** @type {ResponseBody} */
    const responseBody = JSON.parse(decodedResponseBody);
    return responseBody.results;
  } catch (err) {
    if (err instanceof AccessDeniedException) {
      console.error(
```

```
        `Access denied. Ensure you have the correct permissions to invoke
 ${modelId}.`,
      );
    } else {
      throw err;
    }
  }
};

// Invoke the function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const prompt = `Meeting transcript: Miguel: Hi Brant, I want to discuss the
 workstream
    for our new product launch Brant: Sure Miguel, is there anything in particular
 you want
    to discuss? Miguel: Yes, I want to talk about how users enter into the product.
    Brant: Ok, in that case let me add in Namita. Namita: Hey everyone
    Brant: Hi Namita, Miguel wants to discuss how users enter into the product.
    Miguel: its too complicated and we should remove friction.
    for example, why do I need to fill out additional forms?
    I also find it difficult to find where to access the product
    when I first land on the landing page. Brant: I would also add that
    I think there are too many steps. Namita: Ok, I can work on the
    landing page to make the product more discoverable but brant
    can you work on the additonal forms? Brant: Yes but I would need
    to work with James from another team as he needs to unblock the sign up
 workflow.
    Miguel can you document any other concerns so that I can discuss with James only
 once?
    Miguel: Sure.
    From the meeting transcript above, Create a list of action items for each
 person.`;

  console.log("\nModel: Titan Text Express v1");
  console.log(`Prompt: ${prompt}`);

  const results = await invokeTitanTextExpressV1(prompt);
  console.log("Completion:");
  for (const result of results) {
    console.log(result.outputText);
  }
  console.log("\n");
}
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考InvokeModel中的。

使用 Anthropic Claude 2 生成文本

以下代码示例演示了如何通过在 Amazon Bedrock 上调用 Anthropic Claude 2 模型来生成文本。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

调用 Anthropic Claude 2 基础模型以生成文本。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

import { fileURLToPath } from "url";

import {
  AccessDeniedException,
  BedrockRuntimeClient,
  InvokeModelCommand,
} from "@aws-sdk/client-bedrock-runtime";

/**
 * @typedef {Object} ResponseBody
 * @property {string} completion
 */

/**
 * Invokes the Anthropic Claude 2 model to run an inference using the input
 * provided in the request body.
 *
 * @param {string} prompt - The prompt that you want Claude to complete.
 * @returns {string} The inference response (completion) from the model.
 */
export const invokeClaude = async (prompt) => {
```

```
  const client = new BedrockRuntimeClient({ region: "us-east-1" });

  const modelId = "anthropic.claude-v2";

  /* Claude requires you to enclose the prompt as follows: */
  const enclosedPrompt = `Human: ${prompt}\n\nAssistant:`;

  /* The different model providers have individual request and response formats.
   * For the format, ranges, and default values for Anthropic Claude, refer to:
   * https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters-
claude.html
   */
  const payload = {
    prompt: enclosedPrompt,
    max_tokens_to_sample: 500,
    temperature: 0.5,
    stop_sequences: ["\n\nHuman:"],
  };

  const command = new InvokeModelCommand({
    body: JSON.stringify(payload),
    contentType: "application/json",
    accept: "application/json",
    modelId,
  });

  try {
    const response = await client.send(command);
    const decodedResponseBody = new TextDecoder().decode(response.body);

    /** @type {ResponseBody} */
    const responseBody = JSON.parse(decodedResponseBody);

    return responseBody.completion;
  } catch (err) {
    if (err instanceof AccessDeniedException) {
      console.error(
        `Access denied. Ensure you have the correct permissions to invoke
 ${modelId}.`,
      );
    } else {
      throw err;
    }
  }
```

```
};

// Invoke the function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const prompt = 'Complete the following: "Once upon a time..."';
  console.log("\nModel: Anthropic Claude v2");
  console.log(`Prompt: ${prompt}`);

  const completion = await invokeClaude(prompt);
  console.log("Completion:");
  console.log(completion);
  console.log("\n");
}
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考InvokeModel中的。

使用 Meta Llama 2 Chat 生成文本

以下代码示例演示了如何通过在 Amazon Bedrock 上调用 Meta Llama 2 Chat 模型来生成文本。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

调用 Meta Llama 2 Chat 基础模型以生成文本。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

import { fileURLToPath } from "url";

import {
  AccessDeniedException,
  BedrockRuntimeClient,
  InvokeModelCommand,
} from "@aws-sdk/client-bedrock-runtime";
```

```
/**
 * @typedef {Object} ResponseBody
 * @property {generation} text
 */

/**
 * Invokes the Meta Llama 2 Chat model to run an inference
 * using the input provided in the request body.
 *
 * @param {string} prompt - The prompt that you want Llama-2 to complete.
 * @returns {string} The inference response (generation) from the model.
 */
export const invokeLlama2 = async (prompt) => {
  const client = new BedrockRuntimeClient({ region: "us-east-1" });

  const modelId = "meta.llama2-13b-chat-v1";

  /* The different model providers have individual request and response formats.
   * For the format, ranges, and default values for Meta Llama 2 Chat, refer to:
   * https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters-meta.html
   */
  const payload = {
    prompt,
    temperature: 0.5,
    top_p: 0.9,
    max_gen_len: 512,
  };

  const command = new InvokeModelCommand({
    body: JSON.stringify(payload),
    contentType: "application/json",
    accept: "application/json",
    modelId,
  });

  try {
    const response = await client.send(command);
    const decodedResponseBody = new TextDecoder().decode(response.body);

    /** @type {ResponseBody} */
    const responseBody = JSON.parse(decodedResponseBody);

    return responseBody.generation;
  } catch (err) {
```

```
    if (err instanceof AccessDeniedException) {
      console.error(
        `Access denied. Ensure you have the correct permissions to invoke
 ${modelId}.`,
      );
    } else {
      throw err;
    }
  }
};

// Invoke the function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const prompt = 'Complete the following: "Once upon a time..."';
  console.log("\nModel: Meta Llama 2 Chat");
  console.log(`Prompt: ${prompt}`);

  const completion = await invokeLlama2(prompt);
  console.log("Completion:");
  console.log(completion);
  console.log("\n");
}
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考InvokeModel中的。

使用 Mistral 7B 生成文本

以下代码示例显示了如何在 Amazon Bedrock 上调用 Mistral 7B 模型进行文本生成。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

调用 Mistral 7B 基础模型生成文本。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
```

```
import { fileURLToPath } from "url";

import {
  AccessDeniedException,
  BedrockRuntimeClient,
  InvokeModelCommand,
} from "@aws-sdk/client-bedrock-runtime";

/**
 * @typedef {Object} Output
 * @property {string} text
 *
 * @typedef {Object} ResponseBody
 * @property {Output[]} outputs
 */

/**
 * Invokes the Mistral 7B model to run an inference using the input
 * provided in the request body.
 *
 * @param {string} prompt - The prompt that you want Mistral to complete.
 * @returns {string[]} A list of inference responses (completions) from the model.
 */
export const invokeMistral7B = async (prompt) => {
  const client = new BedrockRuntimeClient({ region: "us-west-2" });

  const modelId = "mistral.mistral-7b-instruct-v0:2";

  // Mistral instruct models provide optimal results when embedding
  // the prompt into the following template:
  const instruction = `<s>[INST] ${prompt} [/INST]`;

  const payload = {
    prompt: instruction,
    max_tokens: 500,
    temperature: 0.5,
  };

  const command = new InvokeModelCommand({
    body: JSON.stringify(payload),
    contentType: "application/json",
    accept: "application/json",
    modelId,
```

```
  });

  try {
    const response = await client.send(command);
    const decodedResponseBody = new TextDecoder().decode(response.body);

    /** @type {ResponseBody} */
    const responseBody = JSON.parse(decodedResponseBody);

    return responseBody.outputs.map((output) => output.text);
  } catch (err) {
    if (err instanceof AccessDeniedException) {
      console.error(
        `Access denied. Ensure you have the correct permissions to invoke
 ${modelId}.`,
      );
    } else {
      throw err;
    }
  }
};

// Invoke the function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const prompt = 'Complete the following: "Once upon a time..."';
  console.log("\nModel: Mistral 7B");
  console.log(`Prompt: ${prompt}`);

  const completions = await invokeMistral7B(prompt);
  completions.forEach((completion) => {
    console.log("Completion:");
    console.log(completion);
    console.log("\n");
  });
}
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考InvokeModel中的。

使用 Mixtral 8x7b 生成文本

以下代码示例显示了如何在 Amazon Bedrock 上调用 Mixtral 8x7b 模型模型以生成文本。

适用于 JavaScript (v3) 的软件开发工具包

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

调用 Mixtral 8x7b 基础模型来生成文本。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

import { fileURLToPath } from "url";

import {
  AccessDeniedException,
  BedrockRuntimeClient,
  InvokeModelCommand,
} from "@aws-sdk/client-bedrock-runtime";
import { invokeMistral7B } from "./invoke-mistral7b.js";

/**
 * @typedef {Object} Output
 * @property {string} text
 *
 * @typedef {Object} ResponseBody
 * @property {Output[]} outputs
 */

/**
 * Invokes the Mixtral 8x7B model to run an inference using the input
 * provided in the request body.
 *
 * @param {string} prompt - The prompt that you want Mistral to complete.
 * @returns {string[]} A list of inference responses (completions) from the model.
 */
export const invokeMixtral8x7B = async (prompt) => {
  const client = new BedrockRuntimeClient({ region: "us-west-2" });

  // Mistral instruct models provide optimal results when embedding
  // the prompt into the following template:
  const instruction = `<s>[INST] ${prompt} [/INST]`;
```

```
  const modelId = "mistral.mixtral-8x7b-instruct-v0:1";

  const payload = {
    prompt: instruction,
    max_tokens: 500,
    temperature: 0.5,
  };

  const command = new InvokeModelCommand({
    body: JSON.stringify(payload),
    contentType: "application/json",
    accept: "application/json",
    modelId,
  });

  try {
    const response = await client.send(command);
    const decodedResponseBody = new TextDecoder().decode(response.body);

    /** @type {ResponseBody} */
    const responseBody = JSON.parse(decodedResponseBody);

    return responseBody.outputs.map((output) => output.text);
  } catch (err) {
    if (err instanceof AccessDeniedException) {
      console.error(
        `Access denied. Ensure you have the correct permissions to invoke
 ${modelId}.`,
      );
    } else {
      throw err;
    }
  }
};

// Invoke the function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const prompt = 'Complete the following: "Once upon a time..."';
  console.log("\nModel: Mixtral 8x7B");
  console.log(`Prompt: ${prompt}`);

  const completions = await invokeMistral7B(prompt);
  completions.forEach((completion) => {
```

```
    console.log("Completion:");
    console.log(completion);
    console.log("\n");
  });
}
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考InvokeModel中的。

使用适用于 JavaScript (v3) 的软件开发工具包的 Amazon Bedrock 代理示例

以下代码示例向您展示了如何使用带有 Amazon Bedrock 代理的 Amazon SDK for JavaScript (v3) 来执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景 是展示如何通过在同一服务中调用多个函数来完成特定任务的代码示例。

每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关如何在上下文中设置和运行代码的说明。

开始使用

你好 Amazon Bedrock 的代理人

以下代码示例显示了如何开始使用适用于 Amazon Bedrock 的代理。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

import { fileURLToPath } from "url";

import {
```

```
  BedrockAgentClient,
  GetAgentCommand,
  paginateListAgents,
} from "@aws-sdk/client-bedrock-agent";

/**
 * @typedef {Object} AgentSummary
 */

/**
 * A simple scenario to demonstrate basic setup and interaction with the Bedrock
 Agents Client.
 *
 * This function first initializes the Amazon Bedrock Agents client for a specific
 region.
 * It then retrieves a list of existing agents using the streamlined paginator
 approach.
 * For each agent found, it retrieves detailed information using a command object.
 *
 * Demonstrates:
 * - Use of the Bedrock Agents client to initialize and communicate with the AWS
 service.
 * - Listing resources in a paginated response pattern.
 * - Accessing an individual resource using a command object.
 *
 * @returns {Promise<void>} A promise that resolves when the function has completed
 execution.
 */
export const main = async () => {
  const region = "us-east-1";

  console.log("=".repeat(68));

  console.log(`Initializing Amazon Bedrock Agents client for ${region}...`);
  const client = new BedrockAgentClient({ region });

  console.log(`Retrieving the list of existing agents...`);
  const paginatorConfig = { client };
  const pages = paginateListAgents(paginatorConfig, {});

  /** @type {AgentSummary[]} */
  const agentSummaries = [];
  for await (const page of pages) {
    agentSummaries.push(...page.agentSummaries);
```

```
  }

  console.log(`Found ${agentSummaries.length} agents in ${region}.`);

  if (agentSummaries.length > 0) {
    for (const agentSummary of agentSummaries) {
      const agentId = agentSummary.agentId;
      console.log("=".repeat(68));
      console.log(`Retrieving agent with ID: ${agentId}:`);
      console.log("-".repeat(68));

      const command = new GetAgentCommand({ agentId });
      const response = await client.send(command);
      const agent = response.agent;

      console.log(` Name: ${agent.agentName}`);
      console.log(` Status: ${agent.agentStatus}`);
      console.log(` ARN: ${agent.agentArn}`);
      console.log(` Foundation model: ${agent.foundationModel}`);
    }
  }
  console.log("=".repeat(68));
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  await main();
}
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考中的以下主题。
  - GetAgent
  - ListAgents

主题

- 操作

操作

创建 代理

以下代码示例演示了如何创建 Amazon Bedrock 代理。

适用于 JavaScript (v3) 的软件开发工具包

Note

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

创建 代理

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

import { fileURLToPath } from "url";
import { checkForPlaceholders } from "../lib/utils.js";

import {
  BedrockAgentClient,
  CreateAgentCommand,
} from "@aws-sdk/client-bedrock-agent";

/**
 * Creates an Amazon Bedrock Agent.
 *
 * @param {string} agentName - A name for the agent that you create.
 * @param {string} foundationModel - The foundation model to be used by the agent
 you create.
 * @param {string} agentResourceRoleArn - The ARN of the IAM role with permissions
 required by the agent.
 * @param {string} [region='us-east-1'] - The AWS region in use.
 * @returns {Promise<import("@aws-sdk/client-bedrock-agent").Agent>} An object
 containing details of the created agent.
 */
export const createAgent = async (
  agentName,
  foundationModel,
  agentResourceRoleArn,
```

```
  region = "us-east-1",
) => {
  const client = new BedrockAgentClient({ region });

  const command = new CreateAgentCommand({
    agentName,
    foundationModel,
    agentResourceRoleArn,
  });
  const response = await client.send(command);

  return response.agent;
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  // Replace the placeholders for agentName and accountId, and roleName with a
 unique name for the new agent,
  // the id of your AWS account, and the name of an existing execution role that the
 agent can use inside your account.
  // For foundationModel, specify the desired model. Ensure to remove the brackets
 '[]' before adding your data.

  // A string (max 100 chars) that can include letters, numbers, dashes '-', and
 underscores '_'.
  const agentName = "[your-bedrock-agent-name]";

  // Your AWS account id.
  const accountId = "[123456789012]";

  // The name of the agent's execution role. It must be prefixed by
 `AmazonBedrockExecutionRoleForAgents_`.
  const roleName = "[AmazonBedrockExecutionRoleForAgents_your-role-name]";

  // The ARN for the agent's execution role.
  // Follow the ARN format: 'arn:aws:iam::account-id:role/role-name'
  const roleArn = `arn:aws:iam::${accountId}:role/${roleName}`;

  // Specify the model for the agent. Change if a different model is preferred.
  const foundationModel = "anthropic.claude-v2";

  // Check for unresolved placeholders in agentName and roleArn.
  checkForPlaceholders([agentName, roleArn]);
```

```
  console.log(`Creating a new agent...`);

  const agent = await createAgent(agentName, foundationModel, roleArn);
  console.log(agent);
}
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考CreateAgent中的。

删除代理

以下代码示例演示了如何删除 Amazon Bedrock 代理。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

删除代理。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

import { fileURLToPath } from "url";
import { checkForPlaceholders } from "../lib/utils.js";

import {
  BedrockAgentClient,
  DeleteAgentCommand,
} from "@aws-sdk/client-bedrock-agent";

/**
 * Deletes an Amazon Bedrock Agent.
 *
 * @param {string} agentId - The unique identifier of the agent to delete.
 * @param {string} [region='us-east-1'] - The AWS region in use.
 * @returns {Promise<import("@aws-sdk/client-bedrock-
agent").DeleteAgentCommandOutput>} An object containing the agent id, the status,
 and some additional metadata.
```

```
 */
export const deleteAgent = (agentId, region = "us-east-1") => {
  const client = new BedrockAgentClient({ region });
  const command = new DeleteAgentCommand({ agentId });
  return client.send(command);
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  // Replace the placeholders for agentId with an existing agent's id.
  // Ensure to remove the brackets (`[]`) before adding your data.

  // The agentId must be an alphanumeric string with exactly 10 characters.
  const agentId = "[ABC123DE45]";

  // Check for unresolved placeholders in agentId.
  checkForPlaceholders([agentId]);

  console.log(`Deleting agent with ID ${agentId}...`);

  const response = await deleteAgent(agentId);
  console.log(response);
}
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DeleteAgent中的。

获取代理的相关信息

以下代码示例演示了如何获取有关 Amazon Bedrock 代理的信息。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

获取代理。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
```

```
// SPDX-License-Identifier: Apache-2.0

import { fileURLToPath } from "url";
import { checkForPlaceholders } from "../lib/utils.js";

import {
  BedrockAgentClient,
  GetAgentCommand,
} from "@aws-sdk/client-bedrock-agent";

/**
 * Retrieves the details of an Amazon Bedrock Agent.
 *
 * @param {string} agentId - The unique identifier of the agent.
 * @param {string} [region='us-east-1'] - The AWS region in use.
 * @returns {Promise<import("@aws-sdk/client-bedrock-agent").Agent>} An object
 containing the agent details.
 */
export const getAgent = async (agentId, region = "us-east-1") => {
  const client = new BedrockAgentClient({ region });

  const command = new GetAgentCommand({ agentId });
  const response = await client.send(command);
  return response.agent;
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  // Replace the placeholders for agentId with an existing agent's id.
  // Ensure to remove the brackets '[]' before adding your data.

  // The agentId must be an alphanumeric string with exactly 10 characters.
  const agentId = "[ABC123DE45]";

  // Check for unresolved placeholders in agentId.
  checkForPlaceholders([agentId]);

  console.log(`Retrieving agent with ID ${agentId}...`);

  const agent = await getAgent(agentId);
  console.log(agent);
}
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考GetAgent中的。

列出代理的操作组

以下代码示例演示了如何列出 Amazon Bedrock 代理的操作组。

适用于 JavaScript (v3) 的软件开发工具包

Note

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

列出代理的操作组。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

import { fileURLToPath } from "url";
import { checkForPlaceholders } from "../lib/utils.js";

import {
  BedrockAgentClient,
  ListAgentActionGroupsCommand,
  paginateListAgentActionGroups,
} from "@aws-sdk/client-bedrock-agent";

/**
 * Retrieves a list of Action Groups of an agent utilizing the paginator function.
 *
 * This function leverages a paginator, which abstracts the complexity of
 pagination, providing
 * a straightforward way to handle paginated results inside a `for await...of` loop.
 *
 * @param {string} agentId - The unique identifier of the agent.
 * @param {string} agentVersion - The version of the agent.
 * @param {string} [region='us-east-1'] - The AWS region in use.
 * @returns {Promise<ActionGroupSummary[]>} An array of action group summaries.
 */
export const listAgentActionGroupsWithPaginator = async (
  agentId,
```

```
  agentVersion,
  region = "us-east-1",
) => {
  const client = new BedrockAgentClient({ region });

  // Create a paginator configuration
  const paginatorConfig = {
    client,
    pageSize: 10, // optional, added for demonstration purposes
  };

  const params = { agentId, agentVersion };

  const pages = paginateListAgentActionGroups(paginatorConfig, params);

  // Paginate until there are no more results
  const actionGroupSummaries = [];
  for await (const page of pages) {
    actionGroupSummaries.push(...page.actionGroupSummaries);
  }

  return actionGroupSummaries;
};

/**
 * Retrieves a list of Action Groups of an agent utilizing the
 ListAgentActionGroupsCommand.
 *
 * This function demonstrates the manual approach, sending a command to the client
 and processing the response.
 * Pagination must manually be managed. For a simplified approach that abstracts
 away pagination logic, see
 * the `listAgentActionGroupsWithPaginator()` example below.
 *
 * @param {string} agentId - The unique identifier of the agent.
 * @param {string} agentVersion - The version of the agent.
 * @param {string} [region='us-east-1'] - The AWS region in use.
 * @returns {Promise<ActionGroupSummary[]>} An array of action group summaries.
 */
export const listAgentActionGroupsWithCommandObject = async (
  agentId,
  agentVersion,
  region = "us-east-1",
) => {
```

```
  const client = new BedrockAgentClient({ region });

  let nextToken;
  const actionGroupSummaries = [];
  do {
    const command = new ListAgentActionGroupsCommand({
      agentId,
      agentVersion,
      nextToken,
      maxResults: 10, // optional, added for demonstration purposes
    });

    /** @type {{actionGroupSummaries: ActionGroupSummary[], nextToken?: string}} */
    const response = await client.send(command);

    for (const actionGroup of response.actionGroupSummaries || []) {
      actionGroupSummaries.push(actionGroup);
    }

    nextToken = response.nextToken;
  } while (nextToken);

  return actionGroupSummaries;
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  // Replace the placeholders for agentId and agentVersion with an existing agent's
 id and version.
  // Ensure to remove the brackets '[]' before adding your data.

  // The agentId must be an alphanumeric string with exactly 10 characters.
  const agentId = "[ABC123DE45]";

  // A string either containing `DRAFT` or a number with 1-5 digits (e.g., '123' or
 'DRAFT').
  const agentVersion = "[DRAFT]";

  // Check for unresolved placeholders in agentId and agentVersion.
  checkForPlaceholders([agentId, agentVersion]);

  console.log("=".repeat(68));
  console.log(
    "Listing agent action groups using ListAgentActionGroupsCommand:",
```

```
  );

  for (const actionGroup of await listAgentActionGroupsWithCommandObject(
    agentId,
    agentVersion,
  )) {
    console.log(actionGroup);
  }

  console.log("=".repeat(68));
  console.log(
    "Listing agent action groups using the paginateListAgents function:",
  );
  for (const actionGroup of await listAgentActionGroupsWithPaginator(
    agentId,
    agentVersion,
  )) {
    console.log(actionGroup);
  }
}
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考ListAgentActionGroups中的。

列出可用的代理

以下代码示例演示了如何列出属于某个账户的 Amazon Bedrock 代理。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

列出属于某个账户的代理。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
```

```
import { fileURLToPath } from "url";

import {
  BedrockAgentClient,
  ListAgentsCommand,
  paginateListAgents,
} from "@aws-sdk/client-bedrock-agent";

/**
 * Retrieves a list of available Amazon Bedrock agents utilizing the paginator
 function.
 *
 * This function leverages a paginator, which abstracts the complexity of
 pagination, providing
 * a straightforward way to handle paginated results inside a `for await...of` loop.
 *
 * @param {string} [region='us-east-1'] - The AWS region in use.
 * @returns {Promise<AgentSummary[]>} An array of agent summaries.
 */
export const listAgentsWithPaginator = async (region = "us-east-1") => {
  const client = new BedrockAgentClient({ region });

  const paginatorConfig = {
    client,
    pageSize: 10, // optional, added for demonstration purposes
  };

  const pages = paginateListAgents(paginatorConfig, {});

  // Paginate until there are no more results
  const agentSummaries = [];
  for await (const page of pages) {
    agentSummaries.push(...page.agentSummaries);
  }

  return agentSummaries;
};

/**
 * Retrieves a list of available Amazon Bedrock agents utilizing the
 ListAgentsCommand.
 *
 * This function demonstrates the manual approach, sending a command to the client
 and processing the response.
```

```
 * Pagination must manually be managed. For a simplified approach that abstracts
 away pagination logic, see
 * the `listAgentsWithPaginator()` example below.
 *
 * @param {string} [region='us-east-1'] - The AWS region in use.
 * @returns {Promise<AgentSummary[]>} An array of agent summaries.
 */
export const listAgentsWithCommandObject = async (region = "us-east-1") => {
  const client = new BedrockAgentClient({ region });

  let nextToken;
  const agentSummaries = [];
  do {
    const command = new ListAgentsCommand({
      nextToken,
      maxResults: 10, // optional, added for demonstration purposes
    });

    /** @type {{agentSummaries: AgentSummary[], nextToken?: string}} */
    const paginatedResponse = await client.send(command);

    agentSummaries.push(...(paginatedResponse.agentSummaries || []));

    nextToken = paginatedResponse.nextToken;
  } while (nextToken);

  return agentSummaries;
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  console.log("=".repeat(68));
  console.log("Listing agents using ListAgentsCommand:");
  for (const agent of await listAgentsWithCommandObject()) {
    console.log(agent);
  }

  console.log("=".repeat(68));
  console.log("Listing agents using the paginateListAgents function:");
  for (const agent of await listAgentsWithPaginator()) {
    console.log(agent);
  }
}
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考ListAgents中的。

使用适用于 JavaScript (v3) 的 SDK 的 Amazon Bedrock 运行时代理示例

以下代码示例向您展示了如何使用带有 Amazon Bedrock Runtime 代理的 Amazon SDK for JavaScript (v3) 来执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景 是展示如何通过在同一服务中调用多个函数来完成特定任务的代码示例。

每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关如何在上下文中设置和运行代码的说明。

主题

- 操作

操作

调用代理

以下代码示例演示了如何调用 Amazon Bedrock 代理。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

import {
  BedrockAgentRuntimeClient,
  InvokeAgentCommand,
```

```
} from "@aws-sdk/client-bedrock-agent-runtime";

/**
 * @typedef {Object} ResponseBody
 * @property {string} completion
 */

/**
 * Invokes a Bedrock agent to run an inference using the input
 * provided in the request body.
 *
 * @param {string} prompt - The prompt that you want the Agent to complete.
 * @param {string} sessionId - An arbitrary identifier for the session.
 */
export const invokeBedrockAgent = async (prompt, sessionId) => {
  const client = new BedrockAgentRuntimeClient({ region: "us-east-1" });
  // const client = new BedrockAgentRuntimeClient({
  //   region: "us-east-1",
  //   credentials: {
  //     accessKeyId: "accessKeyId", // permission to invoke agent
  //     secretAccessKey: "accessKeySecret",
  //   },
  // });

  const agentId = "AJBHXXILZN";
  const agentAliasId = "AVKP1ITZAA";

  const command = new InvokeAgentCommand({
    agentId,
    agentAliasId,
    sessionId,
    inputText: prompt,
  });

  try {
    let completion = "";
    const response = await client.send(command);

    if (response.completion === undefined) {
      throw new Error("Completion is undefined");
    }

    for await (let chunkEvent of response.completion) {
      const chunk = chunkEvent.chunk;
```

```
      console.log(chunk);
      const decodedResponse = new TextDecoder("utf-8").decode(chunk.bytes);
      completion += decodedResponse;
    }

    return { sessionId: sessionId, completion };
  } catch (err) {
    console.error(err);
  }
};

// Call function if run directly
import { fileURLToPath } from "url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const result = await invokeBedrockAgent("I need help.", "123");
  console.log(result);
}
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考InvokeAgent中的。

CloudWatch 使用适用于 JavaScript (v3) 的 SDK 的示例

以下代码示例向您展示了如何通过使用 Amazon SDK for JavaScript (v3) 来执行操作和实现常见场景 CloudWatch。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景 是展示如何通过在同一服务中调用多个函数来完成特定任务的代码示例。

每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关如何在上下文中设置和运行代码的说明。

主题

- 操作

操作

创建指标警报

以下代码示例演示如何创建或更新 Amazon CloudWatch 警报，并将其与指定指标、指标数学表达式、异常检测模型或指标见解查询相关联。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

导入 SDK 和客户端模块，然后调用 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();
```

在单独的模块中创建客户端并将其导出。

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考PutMetricAlarm中的。

适用于 JavaScript (v2) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
// 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);
  }
});
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考PutMetricAlarm中的。

删除警报

以下代码示例显示了如何删除 Amazon CloudWatch 警报。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

导入 SDK 和客户端模块，然后调用 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();
```

在单独的模块中创建客户端并将其导出。

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DeleteAlarms中的。

适用于 JavaScript (v2) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

导入 SDK 和客户端模块，然后调用 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);
  }
});
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DeleteAlarms中的。

为指标描述警报

以下代码示例显示了如何描述某个指标的 Amazon CloudWatch 警报。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

导入 SDK 和客户端模块，然后调用 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();
```

在单独的模块中创建客户端并将其导出。

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DescribeAlarmsForMetric中的。

适用于 JavaScript (v2) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
// 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);
    });
  }
});
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DescribeAlarmsForMetric中的。

禁用警报操作

以下代码示例显示了如何禁用 Amazon CloudWatch 警报操作。

适用于 JavaScript (v3) 的软件开发工具包

> Note
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

导入 SDK 和客户端模块，然后调用 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();
```

在单独的模块中创建客户端并将其导出。

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DisableAlarmActions中的。

适用于 JavaScript (v2) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

导入 SDK 和客户端模块，然后调用 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);
    }
  }
);
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DisableAlarmActions中的。

启用警报操作

以下代码示例显示了如何启用 Amazon CloudWatch 警报操作。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

导入 SDK 和客户端模块，然后调用 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();
```

在单独的模块中创建客户端并将其导出。

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";
```

```
export const client = new CloudWatchClient({});
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考EnableAlarmActions中的。

适用于 JavaScript (v2) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

导入 SDK 和客户端模块，然后调用 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);
      }
    });
  }
});
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考EnableAlarmActions中的。

列出指标

以下代码示例显示如何列出 Amazon CloudWatch 指标的元数据。要获取指标的数据，请使用 GetMetricData 或 GetMetricStatistics 操作。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

导入 SDK 和客户端模块，然后调用 API。

```
import { ListMetricsCommand } from "@aws-sdk/client-cloudwatch";
```

```
import { client } from "../libs/client.js";

export const main = () => {
  // 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",
  });

  return client.send(command);
};
```

在单独的模块中创建客户端并将其导出。

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考ListMetrics中的。

适用于 JavaScript (v2) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
// 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));
  }
});
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考ListMetrics中的。

将数据放入指标

以下代码示例展示了如何将指标数据点发布到 Amazon CloudWatch。

适用于 JavaScript (v3) 的软件开发工具包

> Note
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

导入 SDK 和客户端模块，然后调用 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();
```

在单独的模块中创建客户端并将其导出。

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考PutMetricData中的。

适用于 JavaScript (v2) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
// 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));
  }
});
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考PutMetricData中的。

CloudWatch 使用适用于 JavaScript (v3) 的 SDK 的事件示例

以下代码示例向您展示了如何使用 Amazon SDK for JavaScript (v3) with Events 来执行操作和实现常见场景。 CloudWatch

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景 是展示如何通过在同一服务中调用多个函数来完成特定任务的代码示例。

每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关如何在上下文中设置和运行代码的说明。

主题

- 操作

操作

添加目标

以下代码示例显示了如何向 Amazon Events CloudWatch 事件添加目标。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

导入 SDK 和客户端模块，然后调用 API。

```
import { PutTargetsCommand } from "@aws-sdk/client-cloudwatch-events";
import { client } from "../libs/client.js";
```

```
const run = async () => {
  const command = new PutTargetsCommand({
    // The name of the Amazon CloudWatch Events rule.
    Rule: process.env.CLOUDWATCH_EVENTS_RULE,

    // The targets to add to the rule.
    Targets: [
      {
        Arn: process.env.CLOUDWATCH_EVENTS_TARGET_ARN,
        // The ID of the target. Choose a unique ID for each target.
        Id: process.env.CLOUDWATCH_EVENTS_TARGET_ID,
      },
    ],
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

在单独的模块中创建客户端并将其导出。

```
import { CloudWatchEventsClient } from "@aws-sdk/client-cloudwatch-events";

export const client = new CloudWatchEventsClient({});
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考PutTargets中的。

适用于 JavaScript (v2) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatchEvents service object
var cwevents = new AWS.CloudWatchEvents({ apiVersion: "2015-10-07" });

var params = {
  Rule: "DEMO_EVENT",
  Targets: [
    {
      Arn: "LAMBDA_FUNCTION_ARN",
      Id: "myCloudWatchEventsTarget",
    },
  ],
};

cwevents.putTargets(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考PutTargets中的。

创建计划规则

以下代码示例显示了如何创建 Amazon Ev CloudWatch ents 计划规则。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

导入 SDK 和客户端模块，然后调用 API。

```
import { PutRuleCommand } from "@aws-sdk/client-cloudwatch-events";
import { client } from "../libs/client.js";

const run = async () => {
  // Request parameters for PutRule.
  // https://docs.aws.amazon.com/eventbridge/latest/APIReference/
API_PutRule.html#API_PutRule_RequestParameters
  const command = new PutRuleCommand({
    Name: process.env.CLOUDWATCH_EVENTS_RULE,

    // The event pattern for the rule.
    //  Example: {"source": ["my.app"]}
    EventPattern: process.env.CLOUDWATCH_EVENTS_RULE_PATTERN,

    // The state of the rule. Valid values: ENABLED, DISABLED
    State: "ENABLED",
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

在单独的模块中创建客户端并将其导出。

```
import { CloudWatchEventsClient } from "@aws-sdk/client-cloudwatch-events";

export const client = new CloudWatchEventsClient({});
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考PutRule中的。

适用于 JavaScript (v2) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatchEvents service object
var cwevents = new AWS.CloudWatchEvents({ apiVersion: "2015-10-07" });

var params = {
  Name: "DEMO_EVENT",
  RoleArn: "IAM_ROLE_ARN",
  ScheduleExpression: "rate(5 minutes)",
  State: "ENABLED",
};

cwevents.putRule(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.RuleArn);
  }
});
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考PutRule中的。

发送事件

以下代码示例显示了如何发送 Amazon Events CloudWatch 事件。

Note

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

导入 SDK 和客户端模块，然后调用 API。

```
import { PutEventsCommand } from "@aws-sdk/client-cloudwatch-events";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new PutEventsCommand({
    // The list of events to send to Amazon CloudWatch Events.
    Entries: [
      {
        // The name of the application or service that is sending the event.
        Source: "my.app",

        // The name of the event that is being sent.
        DetailType: "My Custom Event",

        // The data that is sent with the event.
        Detail: JSON.stringify({ timeOfEvent: new Date().toISOString() }),
      },
    ],
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

在单独的模块中创建客户端并将其导出。

```
import { CloudWatchEventsClient } from "@aws-sdk/client-cloudwatch-events";

export const client = new CloudWatchEventsClient({});
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考PutEvents中的。

适用于 JavaScript (v2) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatchEvents service object
var cwevents = new AWS.CloudWatchEvents({ apiVersion: "2015-10-07" });

var params = {
  Entries: [
    {
      Detail: '{ "key1": "value1", "key2": "value2" }',
      DetailType: "appRequestSubmitted",
      Resources: ["RESOURCE_ARN"],
      Source: "com.company.app",
    },
  ],
};

cwevents.putEvents(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Entries);
  }
});
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考PutEvents中的。

CloudWatch 使用适用于 JavaScript (v3) 的 SDK 记录示例

以下代码示例向您展示了如何使用带 CloudWatch 日志的 Amazon SDK for JavaScript (v3) 来执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景 是展示如何通过在同一服务中调用多个函数来完成特定任务的代码示例。

每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关如何在上下文中设置和运行代码的说明。

主题

- 操作
- 场景

操作

创建日志组

以下代码示例显示了如何创建新的 CloudWatch 日志日志组。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { CreateLogGroupCommand } from "@aws-sdk/client-cloudwatch-logs";
import { client } from "../libs/client.js";
```

```
const run = async () => {
  const command = new CreateLogGroupCommand({
    // The name of the log group.
    logGroupName: process.env.CLOUDWATCH_LOGS_LOG_GROUP,
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考CreateLogGroup中的。

创建订阅筛选条件

以下代码示例显示了如何创建 Amazon L CloudWatch ogs 订阅筛选条件。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { PutSubscriptionFilterCommand } from "@aws-sdk/client-cloudwatch-logs";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new PutSubscriptionFilterCommand({
    // An ARN of a same-account Kinesis stream, Kinesis Firehose
    // delivery stream, or Lambda function.
    // https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/
SubscriptionFilters.html
    destinationArn: process.env.CLOUDWATCH_LOGS_DESTINATION_ARN,
```

```
    // A name for the filter.
    filterName: process.env.CLOUDWATCH_LOGS_FILTER_NAME,

    // A filter pattern for subscribing to a filtered stream of log events.
    // https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/
FilterAndPatternSyntax.html
    filterPattern: process.env.CLOUDWATCH_LOGS_FILTER_PATTERN,

    // The name of the log group. Messages in this group matching the filter pattern
    // will be sent to the destination ARN.
    logGroupName: process.env.CLOUDWATCH_LOGS_LOG_GROUP,
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考PutSubscriptionFilter中的。

适用于 JavaScript (v2) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the CloudWatchLogs service object
var cwl = new AWS.CloudWatchLogs({ apiVersion: "2014-03-28" });

var params = {
  destinationArn: "LAMBDA_FUNCTION_ARN",
```

```
  filterName: "FILTER_NAME",
  filterPattern: "ERROR",
  logGroupName: "LOG_GROUP",
};

cwl.putSubscriptionFilter(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考PutSubscriptionFilter中的。

删除日志组

以下代码示例显示如何删除现有的 CloudWatch 日志日志组。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { DeleteLogGroupCommand } from "@aws-sdk/client-cloudwatch-logs";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new DeleteLogGroupCommand({
    // The name of the log group.
    logGroupName: process.env.CLOUDWATCH_LOGS_LOG_GROUP,
  });

  try {
    return await client.send(command);
  } catch (err) {
```

```
    console.error(err);
  }
};

export default run();
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DeleteLogGroup中的。

删除订阅筛选条件

以下代码示例显示了如何删除 Amazon Lo CloudWatch gs 订阅筛选条件。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { DeleteSubscriptionFilterCommand } from "@aws-sdk/client-cloudwatch-logs";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new DeleteSubscriptionFilterCommand({
    // The name of the filter.
    filterName: process.env.CLOUDWATCH_LOGS_FILTER_NAME,
    // The name of the log group.
    logGroupName: process.env.CLOUDWATCH_LOGS_LOG_GROUP,
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DeleteSubscriptionFilter中的。

适用于 JavaScript (v2) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the CloudWatchLogs service object
var cwl = new AWS.CloudWatchLogs({ apiVersion: "2014-03-28" });

var params = {
  filterName: "FILTER",
  logGroupName: "LOG_GROUP",
};

cwl.deleteSubscriptionFilter(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DeleteSubscriptionFilter中的。

描述现有订阅筛选条件

以下代码示例说明如何描述 Amazon CloudWatch Logs 现有的订阅筛选条件。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { DescribeSubscriptionFiltersCommand } from "@aws-sdk/client-cloudwatch-
logs";
import { client } from "../libs/client.js";

const run = async () => {
  // This will return a list of all subscription filters in your account
  // matching the log group name.
  const command = new DescribeSubscriptionFiltersCommand({
    logGroupName: process.env.CLOUDWATCH_LOGS_LOG_GROUP,
    limit: 1,
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DescribeSubscriptionFilters中的。

适用于 JavaScript (v2) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the CloudWatchLogs service object
var cwl = new AWS.CloudWatchLogs({ apiVersion: "2014-03-28" });

var params = {
  logGroupName: "GROUP_NAME",
  limit: 5,
};

cwl.describeSubscriptionFilters(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.subscriptionFilters);
  }
});
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DescribeSubscriptionFilters中的。

描述日志组

以下代码示例显示了如何描述 CloudWatch 日志组。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import {
  paginateDescribeLogGroups,
  CloudWatchLogsClient,
```

```
} from "@aws-sdk/client-cloudwatch-logs";

const client = new CloudWatchLogsClient({});

export const main = async () => {
  const paginatedLogGroups = paginateDescribeLogGroups({ client }, {});
  const logGroups = [];

  for await (const page of paginatedLogGroups) {
    if (page.logGroups && page.logGroups.every((lg) => !!lg)) {
      logGroups.push(...page.logGroups);
    }
  }

  console.log(logGroups);
  return logGroups;
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DescribeLogGroups中的。

获取查询的结果

以下代码示例演示了如何获取查询的结果。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
  /**
   * Simple wrapper for the GetQueryResultsCommand.
   * @param {string} queryId
   */
  _getQueryResults(queryId) {
    return this.client.send(new GetQueryResultsCommand({ queryId }));
  }
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考GetQueryResults中的。

开始 Live Tail 会话

以下代码示例演示了如何为现有日志组/日志流启动 Live Tail 会话。

适用于 JavaScript (v3) 的软件开发工具包

包含所需的文件。

```
import { CloudWatchLogsClient, StartLiveTailCommand } from "@aws-sdk/client-
cloudwatch-logs";
```

处理 Live Tail 会话中的事件。

```
async function handleResponseAsync(response) {
    try {
      for await (const event of response.responseStream) {
        if (event.sessionStart !== undefined) {
          console.log(event.sessionStart);
        } else if (event.sessionUpdate !== undefined) {
          for (const logEvent of event.sessionUpdate.sessionResults) {
            const timestamp = logEvent.timestamp;
            const date = new Date(timestamp);
            console.log("[" + date + "] " + logEvent.message);
          }
        } else {
            console.error("Unknown event type");
        }
      }
    } catch (err) {
        // On-stream exceptions are captured here
        console.error(err)
    }
}
```

启动 Live Tail 会话。

```
    const client = new CloudWatchLogsClient();
```

```
    const command = new StartLiveTailCommand({
        logGroupIdentifiers: logGroupIdentifiers,
        logStreamNames: logStreamNames,
        logEventFilterPattern: filterPattern
    });
    try{
        const response = await client.send(command);
        handleResponseAsync(response);
    } catch (err){
        // Pre-stream exceptions are captured here
        console.log(err);
    }
```

经过一段时间后停止 Live Tail 会话。

```
    /* Set a timeout to close the client. This will stop the Live Tail session. */
    setTimeout(function() {
        console.log("Client timeout");
        client.destroy();
      }, 10000);
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考StartLiveTail中的。

开始查询

以下代码示例演示了如何开始查询。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
/**
 * Wrapper for the StartQueryCommand. Uses a static query string
 * for consistency.
 * @param {[Date, Date]} dateRange
```

```
   * @param {number} maxLogs
   * @returns {Promise<{ queryId: string }>}
   */
  async _startQuery([startDate, endDate], maxLogs = 10000) {
    try {
      return await this.client.send(
        new StartQueryCommand({
          logGroupNames: this.logGroupNames,
          queryString: "fields @timestamp, @message | sort @timestamp asc",
          startTime: startDate.valueOf(),
          endTime: endDate.valueOf(),
          limit: maxLogs,
        }),
      );
    } catch (err) {
      /** @type {string} */
      const message = err.message;
      if (message.startsWith("Query's end date and time")) {
        // This error indicates that the query's start or end date occur
        // before the log group was created.
        throw new DateOutOfBoundsError(message);
      }

      throw err;
    }
  }
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考StartQuery中的。

场景

运行大型查询

以下代码示例展示了如何使用 CloudWatch 日志查询超过 10,000 条记录。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

这是入口点。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { CloudWatchLogsClient } from "@aws-sdk/client-cloudwatch-logs";
import { CloudWatchQuery } from "./cloud-watch-query.js";

console.log("Starting a recursive query...");

if (!process.env.QUERY_START_DATE || !process.env.QUERY_END_DATE) {
  throw new Error(
    "QUERY_START_DATE and QUERY_END_DATE environment variables are required.",
  );
}

const cloudWatchQuery = new CloudWatchQuery(new CloudWatchLogsClient({}), {
  logGroupNames: ["/workflows/cloudwatch-logs/large-query"],
  dateRange: [
    new Date(parseInt(process.env.QUERY_START_DATE)),
    new Date(parseInt(process.env.QUERY_END_DATE)),
  ],
});

await cloudWatchQuery.run();

console.log(
  `Queries finished in ${cloudWatchQuery.secondsElapsed} seconds.\nTotal logs found:
 ${cloudWatchQuery.results.length}`,
);
```

该类可在必要时将查询拆分为多个步骤。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
  StartQueryCommand,
  GetQueryResultsCommand,
} from "@aws-sdk/client-cloudwatch-logs";
import { splitDateRange } from "@aws-doc-sdk-examples/lib/utils/util-date.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

class DateOutOfBoundsError extends Error {}
```

```
export class CloudWatchQuery {
  /**
   * Run a query for all CloudWatch Logs within a certain date range.
   * CloudWatch logs return a max of 10,000 results. This class
   * performs a binary search across all of the logs in the provided
   * date range if a query returns the maximum number of results.
   *
   * @param {import('@aws-sdk/client-cloudwatch-logs').CloudWatchLogsClient} client
   * @param {{ logGroupNames: string[], dateRange: [Date, Date], queryConfig:
 { limit: number } }} config
   */
  constructor(client, { logGroupNames, dateRange, queryConfig }) {
    this.client = client;
    /**
     * All log groups are queried.
     */
    this.logGroupNames = logGroupNames;

    /**
     * The inclusive date range that is queried.
     */
    this.dateRange = dateRange;

    /**
     * CloudWatch Logs never returns more than 10,000 logs.
     */
    this.limit = queryConfig?.limit ?? 10000;

    /**
     * @type {import("@aws-sdk/client-cloudwatch-logs").ResultField[][]}
     */
    this.results = [];
  }

  /**
   * Run the query.
   */
  async run() {
    this.secondsElapsed = 0;
    const start = new Date();
    this.results = await this._largeQuery(this.dateRange);
    const end = new Date();
    this.secondsElapsed = (end - start) / 1000;
```

```
    return this.results;
  }

  /**
   * Recursively query for logs.
   * @param {[Date, Date]} dateRange
   * @returns {Promise<import("@aws-sdk/client-cloudwatch-logs").ResultField[][]>}
   */
  async _largeQuery(dateRange) {
    const logs = await this._query(dateRange, this.limit);

    console.log(
      `Query date range: ${dateRange
        .map((d) => d.toISOString())
        .join(" to ")}. Found ${logs.length} logs.`,
    );

    if (logs.length < this.limit) {
      return logs;
    }

    const lastLogDate = this._getLastLogDate(logs);
    const offsetLastLogDate = new Date(lastLogDate);
    offsetLastLogDate.setMilliseconds(lastLogDate.getMilliseconds() + 1);
    const subDateRange = [offsetLastLogDate, dateRange[1]];
    const [r1, r2] = splitDateRange(subDateRange);
    const results = await Promise.all([
      this._largeQuery(r1),
      this._largeQuery(r2),
    ]);
    return [logs, ...results].flat();
  }

  /**
   * Find the most recent log in a list of logs.
   * @param {import("@aws-sdk/client-cloudwatch-logs").ResultField[][]} logs
   */
  _getLastLogDate(logs) {
    const timestamps = logs
      .map(
        (log) =>
          log.find((fieldMeta) => fieldMeta.field === "@timestamp")?.value,
      )
      .filter((t) => !!t)
```

```
      .map((t) => `${t}Z`)
      .sort();

    if (!timestamps.length) {
      throw new Error("No timestamp found in logs.");
    }

    return new Date(timestamps[timestamps.length - 1]);
  }

  // snippet-start:[javascript.v3.cloudwatch-logs.actions.GetQueryResults]
  /**
   * Simple wrapper for the GetQueryResultsCommand.
   * @param {string} queryId
   */
  _getQueryResults(queryId) {
    return this.client.send(new GetQueryResultsCommand({ queryId }));
  }
  // snippet-end:[javascript.v3.cloudwatch-logs.actions.GetQueryResults]

  /**
   * Starts a query and waits for it to complete.
   * @param {[Date, Date]} dateRange
   * @param {number} maxLogs
   */
  async _query(dateRange, maxLogs) {
    try {
      const { queryId } = await this._startQuery(dateRange, maxLogs);
      const { results } = await this._waitUntilQueryDone(queryId);
      return results ?? [];
    } catch (err) {
      /**
       * This error is thrown when StartQuery returns an error indicating
       * that the query's start or end date occur before the log group was
       * created.
       */
      if (err instanceof DateOutOfBoundsError) {
        return [];
      } else {
        throw err;
      }
    }
  }
```

```
  // snippet-start:[javascript.v3.cloudwatch-logs.actions.StartQuery]
  /**
   * Wrapper for the StartQueryCommand. Uses a static query string
   * for consistency.
   * @param {[Date, Date]} dateRange
   * @param {number} maxLogs
   * @returns {Promise<{ queryId: string }>}
   */
  async _startQuery([startDate, endDate], maxLogs = 10000) {
    try {
      return await this.client.send(
        new StartQueryCommand({
          logGroupNames: this.logGroupNames,
          queryString: "fields @timestamp, @message | sort @timestamp asc",
          startTime: startDate.valueOf(),
          endTime: endDate.valueOf(),
          limit: maxLogs,
        }),
      );
    } catch (err) {
      /** @type {string} */
      const message = err.message;
      if (message.startsWith("Query's end date and time")) {
        // This error indicates that the query's start or end date occur
        // before the log group was created.
        throw new DateOutOfBoundsError(message);
      }

      throw err;
    }
  }
  // snippet-end:[javascript.v3.cloudwatch-logs.actions.StartQuery]

  /**
   * Call GetQueryResultsCommand until the query is done.
   * @param {string} queryId
   */
  _waitUntilQueryDone(queryId) {
    const getResults = async () => {
      const results = await this._getQueryResults(queryId);
      const queryDone = [
        "Complete",
        "Failed",
        "Cancelled",
```

```
          "Timeout",
          "Unknown",
        ].includes(results.status);

        return { queryDone, results };
      };

      return retry(
        { intervalInMs: 1000, maxRetries: 60, quiet: true },
        async () => {
          const { queryDone, results } = await getResults();
          if (!queryDone) {
            throw new Error("Query not done.");
          }

          return results;
        },
      );
    }
  }
```

- 有关 API 详细信息，请参阅《Amazon SDK for JavaScript API 参考》中的以下主题。
  - GetQueryResults
  - StartQuery

CodeBuild 使用适用于 JavaScript (v3) 的 SDK 的示例

以下代码示例向您展示了如何通过使用 Amazon SDK for JavaScript (v3) 来执行操作和实现常见场景 CodeBuild。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景 是展示如何通过在同一服务中调用多个函数来完成特定任务的代码示例。

每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关如何在上下文中设置和运行代码的说明。

主题

- 操作

操作

创建项目

以下代码示例显示了如何创建 CodeBuild 项目。

适用于 JavaScript (v3) 的软件开发工具包

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

创建项目。

```
import {
  ArtifactsType,
  CodeBuildClient,
  ComputeType,
  CreateProjectCommand,
  EnvironmentType,
  SourceType,
} from "@aws-sdk/client-codebuild";

// Create the AWS CodeBuild project.
export const createProject = async (
  projectName = "MyCodeBuilder",
  roleArn = "arn:aws:iam::xxxxxxxxxxxx:role/CodeBuildAdmin",
  buildOutputBucket = "xxxx",
  githubUrl = "https://...",
) => {
  const codeBuildClient = new CodeBuildClient({});

  const response = await codeBuildClient.send(
    new CreateProjectCommand({
      artifacts: {
        // The destination of the build artifacts.
        type: ArtifactsType.S3,
        location: buildOutputBucket,
      },
      // Information about the build environment. The combination of "computeType"
 and "type" determines the
```

```
      // requirements for the environment such as CPU, memory, and disk space.
      environment: {
        // Build environment compute types.
        // https://docs.aws.amazon.com/codebuild/latest/userguide/build-env-ref-
compute-types.html
        computeType: ComputeType.BUILD_GENERAL1_SMALL,
        // Docker image identifier.
        // See https://docs.aws.amazon.com/codebuild/latest/userguide/build-env-ref-
available.html
        image: "aws/codebuild/standard:7.0",
        // Build environment type.
        type: EnvironmentType.LINUX_CONTAINER,
      },
      name: projectName,
      // A role ARN with permission to create a CodeBuild project, write to the
 artifact location, and write CloudWatch logs.
      serviceRole: roleArn,
      source: {
        // The type of repository that contains the source code to be built.
        type: SourceType.GITHUB,
        // The location of the repository that contains the source code to be built.
        location: githubUrl,
      },
    }),
  );
  console.log(response);
  //   {
  //     '$metadata': {
  //       httpStatusCode: 200,
  //       requestId: 'b428b244-777b-49a6-a48d-5dffedced8e7',
  //       extendedRequestId: undefined,
  //       cfId: undefined,
  //       attempts: 1,
  //       totalRetryDelay: 0
  //     },
  //     project: {
  //       arn: 'arn:aws:codebuild:us-east-1:xxxxxxxxxxxx:project/MyCodeBuilder',
  //       artifacts: {
  //         encryptionDisabled: false,
  //         location: 'xxxxxx-xxxxxxx-xxxxxx',
  //         name: 'MyCodeBuilder',
  //         namespaceType: 'NONE',
  //         packaging: 'NONE',
  //         type: 'S3'
```

```
  //        },
  //        badge: { badgeEnabled: false },
  //        cache: { type: 'NO_CACHE' },
  //        created: 2023-08-18T14:46:48.979Z,
  //        encryptionKey: 'arn:aws:kms:us-east-1:xxxxxxxxxxxx:alias/aws/s3',
  //        environment: {
  //          computeType: 'BUILD_GENERAL1_SMALL',
  //          environmentVariables: [],
  //          image: 'aws/codebuild/standard:7.0',
  //          imagePullCredentialsType: 'CODEBUILD',
  //          privilegedMode: false,
  //          type: 'LINUX_CONTAINER'
  //        },
  //        lastModified: 2023-08-18T14:46:48.979Z,
  //        name: 'MyCodeBuilder',
  //        projectVisibility: 'PRIVATE',
  //        queuedTimeoutInMinutes: 480,
  //        serviceRole: 'arn:aws:iam::xxxxxxxxxxxx:role/CodeBuildAdmin',
  //        source: {
  //          insecureSsl: false,
  //          location: 'https://...',
  //          reportBuildStatus: false,
  //          type: 'GITHUB'
  //        },
  //        timeoutInMinutes: 60
  //      }
  //    }
  return response;
};
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考CreateProject中的。

使用适用于 JavaScript (v3) 的软件开发工具包的 Amazon Cognito 身份提供商示例

以下代码示例向您展示如何使用带有 Amazon Cognito 身份提供商的 Amazon SDK for JavaScript (v3) 来执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景 是展示如何通过在同一服务中调用多个函数来完成特定任务的代码示例。

每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关如何在上下文中设置和运行代码的说明。

开始使用

开始使用 Amazon Cognito

以下代码示例展示了如何开始使用 Amazon Cognito。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import {
  paginateListUserPools,
  CognitoIdentityProviderClient,
} from "@aws-sdk/client-cognito-identity-provider";

const client = new CognitoIdentityProviderClient({});

export const helloCognito = async () => {
  const paginator = paginateListUserPools({ client }, {});

  const userPoolNames = [];

  for await (const page of paginator) {
    const names = page.UserPools.map((pool) => pool.Name);
    userPoolNames.push(...names);
  }

  console.log("User pool names: ");
  console.log(userPoolNames.join("\n"));
  return userPoolNames;
```

```
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考ListUserPools中的。

主题

- 操作
- 场景

操作

确认用户

以下代码示例展示了如何确认 Amazon Cognito 用户。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
const confirmSignUp = ({ clientId, username, code }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new ConfirmSignUpCommand({
    ClientId: clientId,
    Username: username,
    ConfirmationCode: code,
  });

  return client.send(command);
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考ConfirmSignUp中的。

确认 MFA 设备可供跟踪

以下代码示例展示了如何确认 MFA 设备可通过 Amazon Cognito 进行跟踪。

适用于 JavaScript (v3) 的软件开发工具包

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
const confirmDevice = ({ deviceKey, accessToken, passwordVerifier, salt }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new ConfirmDeviceCommand({
    DeviceKey: deviceKey,
    AccessToken: accessToken,
    DeviceSecretVerifierConfig: {
      PasswordVerifier: passwordVerifier,
      Salt: salt,
    },
  });

  return client.send(command);
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考ConfirmDevice中的。

获取令牌以将 MFA 应用程序与用户关联

以下代码示例展示了如何获取令牌，以将 MFA 应用程序与 Amazon Cognito 用户关联。

适用于 JavaScript (v3) 的软件开发工具包

Note

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
const associateSoftwareToken = (session) => {
  const client = new CognitoIdentityProviderClient({});
  const command = new AssociateSoftwareTokenCommand({
    Session: session,
  });

  return client.send(command);
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考AssociateSoftwareToken中的。

获取有关用户的信息

以下代码示例展示了如何获取有关 Amazon Cognito 用户的信息。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
const adminGetUser = ({ userPoolId, username }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new AdminGetUserCommand({
    UserPoolId: userPoolId,
    Username: username,
  });

  return client.send(command);
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考AdminGetUser中的。

列出用户

以下代码示例展示了如何列出 Amazon Cognito 用户。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
const listUsers = ({ userPoolId }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new ListUsersCommand({
    UserPoolId: userPoolId,
  });

  return client.send(command);
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考ListUsers中的。

重新发送确认码

以下代码示例展示了如何重新发送 Amazon Cognito 确认码。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
const resendConfirmationCode = ({ clientId, username }) => {
  const client = new CognitoIdentityProviderClient({});
```

```
  const command = new ResendConfirmationCodeCommand({
    ClientId: clientId,
    Username: username,
  });

  return client.send(command);
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考ResendConfirmationCode中的。

响应 SRP 身份验证质询

以下代码示例展示了如何响应 Amazon Cognito SRP 身份验证质询。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
const respondToAuthChallenge = ({
  clientId,
  username,
  session,
  userPoolId,
  code,
}) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new RespondToAuthChallengeCommand({
    ChallengeName: ChallengeNameType.SOFTWARE_TOKEN_MFA,
    ChallengeResponses: {
      SOFTWARE_TOKEN_MFA_CODE: code,
      USERNAME: username,
    },
    ClientId: clientId,
```

```
    UserPoolId: userPoolId,
    Session: session,
  });

  return client.send(command);
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考RespondToAuthChallenge中的。

响应身份验证质询

以下代码示例展示了如何响应 Amazon Cognito 身份验证质询。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
const adminRespondToAuthChallenge = ({
  userPoolId,
  clientId,
  username,
  totp,
  session,
}) => {
  const client = new CognitoIdentityProviderClient({});
  const command = new AdminRespondToAuthChallengeCommand({
    ChallengeName: ChallengeNameType.SOFTWARE_TOKEN_MFA,
    ChallengeResponses: {
      SOFTWARE_TOKEN_MFA_CODE: totp,
      USERNAME: username,
    },
    ClientId: clientId,
    UserPoolId: userPoolId,
    Session: session,
  });
```

```
  return client.send(command);
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考AdminRespondToAuthChallenge中的。

注册用户

以下代码示例展示了如何向 Amazon Cognito 注册用户。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
const signUp = ({ clientId, username, password, email }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new SignUpCommand({
    ClientId: clientId,
    Username: username,
    Password: password,
    UserAttributes: [{ Name: "email", Value: email }],
  });

  return client.send(command);
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考SignUp中的。

开始身份验证

以下代码示例演示了如何通过 Amazon Cognito 开始身份验证。

适用于 JavaScript (v3) 的软件开发工具包

Note

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
const initiateAuth = ({ username, password, clientId }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new InitiateAuthCommand({
    AuthFlow: AuthFlowType.USER_PASSWORD_AUTH,
    AuthParameters: {
      USERNAME: username,
      PASSWORD: password,
    },
    ClientId: clientId,
  });

  return client.send(command);
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考InitiateAuth中的。

使用管理员凭证开始身份验证

以下代码示例展示了如何使用 Amazon Cognito 和管理员凭证开始身份验证。

适用于 JavaScript (v3) 的软件开发工具包

Note

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
const adminInitiateAuth = ({ clientId, userPoolId, username, password }) => {
  const client = new CognitoIdentityProviderClient({});
```

```
  const command = new AdminInitiateAuthCommand({
    ClientId: clientId,
    UserPoolId: userPoolId,
    AuthFlow: AuthFlowType.ADMIN_USER_PASSWORD_AUTH,
    AuthParameters: { USERNAME: username, PASSWORD: password },
  });

  return client.send(command);
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考AdminInitiateAuth中的。

向用户验证 MFA 应用程序

以下代码示例展示了如何向 Amazon Cognito 用户验证 MFA 应用程序。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
const verifySoftwareToken = (totp) => {
  const client = new CognitoIdentityProviderClient({});

  // The 'Session' is provided in the response to 'AssociateSoftwareToken'.
  const session = process.env.SESSION;

  if (!session) {
    throw new Error(
      "Missing a valid Session. Did you run 'admin-initiate-auth'?",
    );
  }

  const command = new VerifySoftwareTokenCommand({
    Session: session,
    UserCode: totp,
  });
```

```
  return client.send(command);
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考VerifySoftwareToken中的。

场景

向需要 MFA 的用户池注册用户

以下代码示例展示了如何：

- 使用用户名、密码和电子邮件地址注册和确认用户。
- 通过将 MFA 应用程序与用户关联来设置多重身份验证。
- 使用密码和 MFA 代码登录。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

为了获得最佳体验，请克隆 GitHub 存储库并运行此示例。以下代码代表完整示例应用程序的示例。

```
import { log } from "@aws-doc-sdk-examples/lib/utils/util-log.js";
import { signUp } from "../../../actions/sign-up.js";
import { FILE_USER_POOLS } from "./constants.js";
import { getSecondValuesFromEntries } from "@aws-doc-sdk-examples/lib/utils/util-
csv.js";

const validateClient = (clientId) => {
  if (!clientId) {
    throw new Error(
      `App client id is missing. Did you run 'create-user-pool'?`,
    );
  }
```

```
};

const validateUser = (username, password, email) => {
  if (!(username && password && email)) {
    throw new Error(
      `Username, password, and email must be provided as arguments to the 'sign-up'
 command.`,
    );
  }
};

const signUpHandler = async (commands) => {
  const [_, username, password, email] = commands;

  try {
    validateUser(username, password, email);
    /**
     * @type {string[]}
     */
    const values = getSecondValuesFromEntries(FILE_USER_POOLS);
    const clientId = values[0];
    validateClient(clientId);
    log(`Signing up.`);
    await signUp({ clientId, username, password, email });
    log(`Signed up. A confirmation email has been sent to: ${email}.`);
    log(`Run 'confirm-sign-up ${username} <code>' to confirm your account.`);
  } catch (err) {
    log(err);
  }
};

export { signUpHandler };

const signUp = ({ clientId, username, password, email }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new SignUpCommand({
    ClientId: clientId,
    Username: username,
    Password: password,
    UserAttributes: [{ Name: "email", Value: email }],
  });

  return client.send(command);
```

```
};

import { log } from "@aws-doc-sdk-examples/lib/utils/util-log.js";
import { confirmSignUp } from "../../../actions/confirm-sign-up.js";
import { FILE_USER_POOLS } from "./constants.js";
import { getSecondValuesFromEntries } from "@aws-doc-sdk-examples/lib/utils/util-
csv.js";

const validateClient = (clientId) => {
  if (!clientId) {
    throw new Error(
      `App client id is missing. Did you run 'create-user-pool'?`,
    );
  }
};

const validateUser = (username) => {
  if (!username) {
    throw new Error(
      `Username name is missing. It must be provided as an argument to the 'confirm-
sign-up' command.`,
    );
  }
};

const validateCode = (code) => {
  if (!code) {
    throw new Error(
      `Verification code is missing. It must be provided as an argument to the
 'confirm-sign-up' command.`,
    );
  }
};

const confirmSignUpHandler = async (commands) => {
  const [_, username, code] = commands;

  try {
    validateUser(username);
    validateCode(code);
    /**
     * @type {string[]}
     */
    const values = getSecondValuesFromEntries(FILE_USER_POOLS);
```

```
    const clientId = values[0];
    validateClient(clientId);
    log(`Confirming user.`);
    await confirmSignUp({ clientId, username, code });
    log(
      `User confirmed. Run 'admin-initiate-auth ${username} <password>' to sign
 in.`,
    );
  } catch (err) {
    log(err);
  }
};

export { confirmSignUpHandler };

const confirmSignUp = ({ clientId, username, code }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new ConfirmSignUpCommand({
    ClientId: clientId,
    Username: username,
    ConfirmationCode: code,
  });

  return client.send(command);
};

import qrcode from "qrcode-terminal";
import { log } from "@aws-doc-sdk-examples/lib/utils/util-log.js";
import { adminInitiateAuth } from "../../../actions/admin-initiate-auth.js";
import { associateSoftwareToken } from "../../../actions/associate-software-
token.js";
import { FILE_USER_POOLS } from "./constants.js";
import { getFirstEntry } from "@aws-doc-sdk-examples/lib/utils/util-csv.js";

const handleMfaSetup = async (session, username) => {
  const { SecretCode, Session } = await associateSoftwareToken(session);

  // Store the Session for use with 'VerifySoftwareToken'.
  process.env.SESSION = Session;

  console.log(
    "Scan this code in your preferred authenticator app, then run 'verify-software-
token' to finish the setup.",
```

```
  );
  qrcode.generate(
    `otpauth://totp/${username}?secret=${SecretCode}`,
    { small: true },
    console.log,
  );
};

const handleSoftwareTokenMfa = (session) => {
  // Store the Session for use with 'AdminRespondToAuthChallenge'.
  process.env.SESSION = session;
};

const validateClient = (id) => {
  if (!id) {
    throw new Error(
      `User pool client id is missing. Did you run 'create-user-pool'?`,
    );
  }
};

const validateId = (id) => {
  if (!id) {
    throw new Error(`User pool id is missing. Did you run 'create-user-pool'?`);
  }
};

const validateUser = (username, password) => {
  if (!(username && password)) {
    throw new Error(
      `Username and password must be provided as arguments to the 'admin-initiate-
auth' command.`,
    );
  }
};

const adminInitiateAuthHandler = async (commands) => {
  const [_, username, password] = commands;

  try {
    validateUser(username, password);

    const [userPoolId, clientId] = getFirstEntry(FILE_USER_POOLS);
    validateId(userPoolId);
```

```
    validateClient(clientId);

    log("Signing in.");
    const { ChallengeName, Session } = await adminInitiateAuth({
      clientId,
      userPoolId,
      username,
      password,
    });

    if (ChallengeName === "MFA_SETUP") {
      log("MFA setup is required.");
      return handleMfaSetup(Session, username);
    }

    if (ChallengeName === "SOFTWARE_TOKEN_MFA") {
      handleSoftwareTokenMfa(Session);
      log(`Run 'admin-respond-to-auth-challenge ${username} <totp>'`);
    }
  } catch (err) {
    log(err);
  }
};

export { adminInitiateAuthHandler };

const adminInitiateAuth = ({ clientId, userPoolId, username, password }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new AdminInitiateAuthCommand({
    ClientId: clientId,
    UserPoolId: userPoolId,
    AuthFlow: AuthFlowType.ADMIN_USER_PASSWORD_AUTH,
    AuthParameters: { USERNAME: username, PASSWORD: password },
  });

  return client.send(command);
};

import { log } from "@aws-doc-sdk-examples/lib/utils/util-log.js";
import { adminRespondToAuthChallenge } from "../../../actions/admin-respond-to-auth-
challenge.js";
import { getFirstEntry } from "@aws-doc-sdk-examples/lib/utils/util-csv.js";
import { FILE_USER_POOLS } from "./constants.js";
```

```
const verifyUsername = (username) => {
  if (!username) {
    throw new Error(
      `Username is missing. It must be provided as an argument to the 'admin-respond-to-auth-challenge' command.`,
    );
  }
};

const verifyTotp = (totp) => {
  if (!totp) {
    throw new Error(
      `Time-based one-time password (TOTP) is missing. It must be provided as an argument to the 'admin-respond-to-auth-challenge' command.`,
    );
  }
};

const storeAccessToken = (token) => {
  process.env.AccessToken = token;
};

const adminRespondToAuthChallengeHandler = async (commands) => {
  const [_, username, totp] = commands;

  try {
    verifyUsername(username);
    verifyTotp(totp);

    const [userPoolId, clientId] = getFirstEntry(FILE_USER_POOLS);
    const session = process.env.SESSION;

    const { AuthenticationResult } = await adminRespondToAuthChallenge({
      clientId,
      userPoolId,
      username,
      totp,
      session,
    });

    storeAccessToken(AuthenticationResult.AccessToken);

    log("Successfully authenticated.");
```

```
  } catch (err) {
    log(err);
  }
};

export { adminRespondToAuthChallengeHandler };

const respondToAuthChallenge = ({
  clientId,
  username,
  session,
  userPoolId,
  code,
}) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new RespondToAuthChallengeCommand({
    ChallengeName: ChallengeNameType.SOFTWARE_TOKEN_MFA,
    ChallengeResponses: {
      SOFTWARE_TOKEN_MFA_CODE: code,
      USERNAME: username,
    },
    ClientId: clientId,
    UserPoolId: userPoolId,
    Session: session,
  });

  return client.send(command);
};

import { log } from "@aws-doc-sdk-examples/lib/utils/util-log.js";
import { verifySoftwareToken } from "../../../actions/verify-software-token.js";

const validateTotp = (totp) => {
  if (!totp) {
    throw new Error(
      `Time-based one-time password (TOTP) must be provided to the 'validate-
software-token' command.`,
    );
  }
};
const verifySoftwareTokenHandler = async (commands) => {
  const [_, totp] = commands;
```

```
  try {
    validateTotp(totp);

    log("Verifying TOTP.");
    await verifySoftwareToken(totp);
    log("TOTP Verified. Run 'admin-initiate-auth' again to sign-in.");
  } catch (err) {
    console.log(err);
  }
};

export { verifySoftwareTokenHandler };

const verifySoftwareToken = (totp) => {
  const client = new CognitoIdentityProviderClient({});

  // The 'Session' is provided in the response to 'AssociateSoftwareToken'.
  const session = process.env.SESSION;

  if (!session) {
    throw new Error(
      "Missing a valid Session. Did you run 'admin-initiate-auth'?",
    );
  }

  const command = new VerifySoftwareTokenCommand({
    Session: session,
    UserCode: totp,
  });

  return client.send(command);
};
```

- 有关 API 详细信息，请参阅《Amazon SDK for JavaScript API 参考》中的以下主题。
  - AdminGetUser
  - AdminInitiateAuth
  - AdminRespondToAuthChallenge
  - AssociateSoftwareToken
  - ConfirmDevice
  - ConfirmSignUp

- InitiateAuth
- ListUsers
- ResendConfirmationCode
- RespondToAuthChallenge
- SignUp
- VerifySoftwareToken

使用适用于 (v3) 的 SDK JavaScript 的 DynamoDB 示例

以下代码示例向您展示了如何使用带有 DynamoDB 的 Amazon SDK for JavaScript (v3) 来执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景 是展示如何通过在同一服务中调用多个函数来完成特定任务的代码示例。

每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关如何在上下文中设置和运行代码的说明。

开始使用

开始使用 DynamoDB

以下代码示例展示了如何开始使用 DynamoDB。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { ListTablesCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
```

```
  const command = new ListTablesCommand({});

  const response = await client.send(command);
  console.log(response.TableNames.join("\n"));
  return response;
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考ListTables中的。

主题

- 操作
- 场景

操作

创建表

以下代码示例展示了如何创建 DynamoDB 表。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { CreateTableCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new CreateTableCommand({
    TableName: "EspressoDrinks",
    // For more information about data types,
    // see https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
HowItWorks.NamingRulesDataTypes.html#HowItWorks.DataTypes and
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
```

```
    AttributeDefinitions: [
      {
        AttributeName: "DrinkName",
        AttributeType: "S",
      },
    ],
    KeySchema: [
      {
        AttributeName: "DrinkName",
        KeyType: "HASH",
      },
    ],
    ProvisionedThroughput: {
      ReadCapacityUnits: 1,
      WriteCapacityUnits: 1,
    },
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考CreateTable中的。

适用于 JavaScript (v2) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });
```

```
var params = {
  AttributeDefinitions: [
    {
      AttributeName: "CUSTOMER_ID",
      AttributeType: "N",
    },
    {
      AttributeName: "CUSTOMER_NAME",
      AttributeType: "S",
    },
  ],
  KeySchema: [
    {
      AttributeName: "CUSTOMER_ID",
      KeyType: "HASH",
    },
    {
      AttributeName: "CUSTOMER_NAME",
      KeyType: "RANGE",
    },
  ],
  ProvisionedThroughput: {
    ReadCapacityUnits: 1,
    WriteCapacityUnits: 1,
  },
  TableName: "CUSTOMER_LIST",
  StreamSpecification: {
    StreamEnabled: false,
  },
};

// Call DynamoDB to create the table
ddb.createTable(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Table Created", data);
  }
});
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考CreateTable中的。

删除表

以下代码示例展示了如何删除 DynamoDB 表。

适用于 JavaScript (v3) 的软件开发工具包

> Note
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { DeleteTableCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new DeleteTableCommand({
    TableName: "DecafCoffees",
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DeleteTable中的。

适用于 JavaScript (v2) 的软件开发工具包

> Note
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
```

```
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: process.argv[2],
};

// Call DynamoDB to delete the specified table
ddb.deleteTable(params, function (err, data) {
  if (err && err.code === "ResourceNotFoundException") {
    console.log("Error: Table not found");
  } else if (err && err.code === "ResourceInUseException") {
    console.log("Error: Table in use");
  } else {
    console.log("Success", data);
  }
});
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DeleteTable中的。

删除表中项目

以下代码示例展示了如何从 DynamoDB 表中删除项目。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

此示例使用文档客户端来简化在 DynamoDB 中处理项目的过程。有关 API 的详细信息，请参阅DeleteCommand。

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
```

```
import { DynamoDBDocumentClient, DeleteCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new DeleteCommand({
    TableName: "Sodas",
    Key: {
      Flavor: "Cola",
    },
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DeleteItem中的。

适用于 JavaScript (v2) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

从表中删除项目。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: "TABLE",
```

```
  Key: {
    KEY_NAME: { N: "VALUE" },
  },
};

// Call DynamoDB to delete the item from the table
ddb.deleteItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

使用 DynamoDB 文档客户端从表中删除项目。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  Key: {
    HASH_KEY: VALUE,
  },
  TableName: "TABLE",
};

docClient.delete(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DeleteItem中的。

获取一批项目

以下代码示例展示了如何获取一批 DynamoDB 项目。

适用于 JavaScript (v3) 的软件开发工具包

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

此示例使用文档客户端来简化在 DynamoDB 中处理项目的过程。有关 API 的详细信息，请参阅BatchGet。

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { BatchGetCommand, DynamoDBDocumentClient } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new BatchGetCommand({
    // Each key in this object is the name of a table. This example refers
    // to a Books table.
    RequestItems: {
      Books: {
        // Each entry in Keys is an object that specifies a primary key.
        Keys: [
          {
            Title: "How to AWS",
          },
          {
            Title: "DynamoDB for DBAs",
          },
        ],
        // Only return the "Title" and "PageCount" attributes.
        ProjectionExpression: "Title, PageCount",
      },
    },
  });

  const response = await docClient.send(command);
```

```
  console.log(response.Responses["Books"]);
  return response;
};
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考BatchGetItem中的。

适用于 JavaScript (v2) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  RequestItems: {
    TABLE_NAME: {
      Keys: [
        { KEY_NAME: { N: "KEY_VALUE_1" } },
        { KEY_NAME: { N: "KEY_VALUE_2" } },
        { KEY_NAME: { N: "KEY_VALUE_3" } },
      ],
      ProjectionExpression: "KEY_NAME, ATTRIBUTE",
    },
  },
};

ddb.batchGetItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    data.Responses.TABLE_NAME.forEach(function (element, index, array) {
```

```
      console.log(element);
    });
  }
});
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考BatchGetItem中的。

获取表中项目

以下代码示例展示了如何从 DynamoDB 表中获取项目。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

此示例使用文档客户端来简化在 DynamoDB 中处理项目的过程。有关 API 的详细信息，请参阅GetCommand。

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, GetCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new GetCommand({
    TableName: "AngryAnimals",
    Key: {
      CommonName: "Shoebill",
    },
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
```

```
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考GetItem中的。

适用于 JavaScript (v2) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

从表中获取项目。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: "TABLE",
  Key: {
    KEY_NAME: { N: "001" },
  },
  ProjectionExpression: "ATTRIBUTE_NAME",
};

// Call DynamoDB to read the item from the table
ddb.getItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Item);
  }
});
```

使用 DynamoDB 文档客户端从表中获取项目。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  TableName: "EPISODES_TABLE",
  Key: { KEY_NAME: VALUE },
};

docClient.get(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Item);
  }
});
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考GetItem中的。

获取表信息

以下代码示例展示了如何获取有关 DynamoDB 表的信息。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { DescribeTableCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});
```

```
export const main = async () => {
  const command = new DescribeTableCommand({
    TableName: "Pastries",
  });

  const response = await client.send(command);
  console.log(`TABLE NAME: ${response.Table.TableName}`);
  console.log(`TABLE ITEM COUNT: ${response.Table.ItemCount}`);
  return response;
};
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DescribeTable中的。

适用于 JavaScript (v2) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: process.argv[2],
};

// Call DynamoDB to retrieve the selected table descriptions
ddb.describeTable(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Table.KeySchema);
  }
```

```
});
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DescribeTable中的。

列出表

以下代码示例展示了如何列出 DynamoDB 表。

适用于 JavaScript (v3) 的软件开发工具包

> Note
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { ListTablesCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new ListTablesCommand({});

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考ListTables中的。

适用于 JavaScript (v2) 的软件开发工具包

> Note
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

// Call DynamoDB to retrieve the list of tables
ddb.listTables({ Limit: 10 }, function (err, data) {
  if (err) {
    console.log("Error", err.code);
  } else {
    console.log("Table names are ", data.TableNames);
  }
});
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考ListTables中的。

将项目放入表中

以下代码示例展示了如何将项目放入 DynamoDB 表中。

适用于 JavaScript (v3) 的软件开发工具包

Note

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

此示例使用文档客户端来简化在 DynamoDB 中处理项目的过程。有关 API 的详细信息，请参阅PutCommand。

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { PutCommand, DynamoDBDocumentClient } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);
```

```
export const main = async () => {
  const command = new PutCommand({
    TableName: "HappyAnimals",
    Item: {
      CommonName: "Shiba Inu",
    },
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考PutItem中的。

适用于 JavaScript (v2) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

将项目放入表中。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: "CUSTOMER_LIST",
  Item: {
    CUSTOMER_ID: { N: "001" },
    CUSTOMER_NAME: { S: "Richard Roe" },
  },
};
```

```
// Call DynamoDB to add the item to the table
ddb.putItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

使用 DynamoDB 文档客户端将项目放入表中。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  TableName: "TABLE",
  Item: {
    HASHKEY: VALUE,
    ATTRIBUTE_1: "STRING_VALUE",
    ATTRIBUTE_2: VALUE_2,
  },
};

docClient.put(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考PutItem中的。

查询表

以下代码示例展示了如何查询 DynamoDB 表。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

此示例使用文档客户端来简化在 DynamoDB 中处理项目的过程。有关 API 的详细信息，请参阅QueryCommand。

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { QueryCommand, DynamoDBDocumentClient } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new QueryCommand({
    TableName: "CoffeeCrop",
    KeyConditionExpression:
      "OriginCountry = :originCountry AND RoastDate > :roastDate",
    ExpressionAttributeValues: {
      ":originCountry": "Ethiopia",
      ":roastDate": "2023-05-01",
    },
    ConsistentRead: true,
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 详细信息，请参阅《Amazon SDK for JavaScript API 参考》中的 Query。

适用于 JavaScript (v2) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  ExpressionAttributeValues: {
    ":s": 2,
    ":e": 9,
    ":topic": "PHRASE",
  },
  KeyConditionExpression: "Season = :s and Episode > :e",
  FilterExpression: "contains (Subtitle, :topic)",
  TableName: "EPISODES_TABLE",
};

docClient.query(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Items);
  }
});
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考中的 Query。

运行 PartiQL 语句

以下代码示例展示了如何在 DynamoDB 表上运行 PartiQL 语句。

适用于 JavaScript (v3) 的软件开发工具包

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

使用 PartiQL 创建项目。

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  ExecuteStatementCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new ExecuteStatementCommand({
    Statement: `INSERT INTO Flowers value {'Name':?}`,
    Parameters: ["Rose"],
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

使用 PartiQL 获取项目。

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  ExecuteStatementCommand,
```

```
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new ExecuteStatementCommand({
    Statement: "SELECT * FROM CloudTypes WHERE IsStorm=?",
    Parameters: [false],
    ConsistentRead: true,
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

使用 PartiQL 更新项目。

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  ExecuteStatementCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new ExecuteStatementCommand({
    Statement: "UPDATE EyeColors SET IsRecessive=? where Color=?",
    Parameters: [true, "blue"],
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

使用 PartiQL 删除项目。

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  ExecuteStatementCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new ExecuteStatementCommand({
    Statement: "DELETE FROM PaintColors where Name=?",
    Parameters: ["Purple"],
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考ExecuteStatement中的。

运行批量 PartiQL 语句

以下代码示例展示了如何在 DynamoDB 表中运行批量 PartiQL 语句。

适用于 JavaScript (v3) 的软件开发工具包

> Note
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

使用 PartiQL 创建一批项目。

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
```

```
import {
  DynamoDBDocumentClient,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const breakfastFoods = ["Eggs", "Bacon", "Sausage"];
  const command = new BatchExecuteStatementCommand({
    Statements: breakfastFoods.map((food) => ({
      Statement: `INSERT INTO BreakfastFoods value {'Name':?}`,
      Parameters: [food],
    })),
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

使用 PartiQL 获取一批项目。

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  DynamoDBDocumentClient,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new BatchExecuteStatementCommand({
    Statements: [
      {
        Statement: "SELECT * FROM PepperMeasurements WHERE Unit=?",
        Parameters: ["Teaspoons"],
        ConsistentRead: true,
```

```
      },
      {
        Statement: "SELECT * FROM PepperMeasurements WHERE Unit=?",
        Parameters: ["Grams"],
        ConsistentRead: true,
      },
    ],
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

使用 PartiQL 更新一批项目。

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  DynamoDBDocumentClient,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const eggUpdates = [
    ["duck", "fried"],
    ["chicken", "omelette"],
  ];
  const command = new BatchExecuteStatementCommand({
    Statements: eggUpdates.map((change) => ({
      Statement: "UPDATE Eggs SET Style=? where Variety=?",
      Parameters: [change[1], change[0]],
    })),
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

使用 PartiQL 删除一批项目。

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  DynamoDBDocumentClient,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new BatchExecuteStatementCommand({
    Statements: [
      {
        Statement: "DELETE FROM Flavors where Name=?",
        Parameters: ["Grape"],
      },
      {
        Statement: "DELETE FROM Flavors where Name=?",
        Parameters: ["Strawberry"],
      },
    ],
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考BatchExecuteStatement中的。

扫描表

以下代码示例展示了如何扫描 DynamoDB 表。

适用于 JavaScript (v3) 的软件开发工具包

Note

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

此示例使用文档客户端来简化在 DynamoDB 中处理项目的过程。有关 API 的详细信息，请参阅ScanCommand。

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, ScanCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new ScanCommand({
    ProjectionExpression: "#Name, Color, AvgLifeSpan",
    ExpressionAttributeNames: { "#Name": "Name" },
    TableName: "Birds",
  });

  const response = await docClient.send(command);
  for (const bird of response.Items) {
    console.log(`${bird.Name} - (${bird.Color}, ${bird.AvgLifeSpan})`);
  }
  return response;
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考中的 Scan。

适用于 JavaScript (v2) 的软件开发工具包

Note

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
// Load the AWS SDK for Node.js.
var AWS = require("aws-sdk");
// Set the AWS Region.
AWS.config.update({ region: "REGION" });

// Create DynamoDB service object.
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

const params = {
  // Specify which items in the results are returned.
  FilterExpression: "Subtitle = :topic AND Season = :s AND Episode = :e",
  // Define the expression attribute value, which are substitutes for the values you
 want to compare.
  ExpressionAttributeValues: {
    ":topic": { S: "SubTitle2" },
    ":s": { N: 1 },
    ":e": { N: 2 },
  },
  // Set the projection expression, which are the attributes that you want.
  ProjectionExpression: "Season, Episode, Title, Subtitle",
  TableName: "EPISODES_TABLE",
};

ddb.scan(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
    data.Items.forEach(function (element, index, array) {
      console.log(
        "printing",
        element.Title.S + " (" + element.Subtitle.S + ")"
      );
    });
  }
});
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 详细信息，请参阅 Amazon SDK for JavaScript API 参考中的 Scan。

更新表中项目

以下代码示例展示了如何更新 DynamoDB 表中的项目。

适用于 JavaScript (v3) 的软件开发工具包

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

此示例使用文档客户端来简化在 DynamoDB 中处理项目的过程。有关 API 的详细信息，请参阅UpdateCommand。

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, UpdateCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new UpdateCommand({
    TableName: "Dogs",
    Key: {
      Breed: "Labrador",
    },
    UpdateExpression: "set Color = :color",
    ExpressionAttributeValues: {
      ":color": "black",
    },
    ReturnValues: "ALL_NEW",
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考UpdateItem中的。

写入一批项目

以下代码示例展示了如何写入一批 DynamoDB 项目。

适用于 JavaScript (v3) 的软件开发工具包

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

此示例使用文档客户端来简化在 DynamoDB 中处理项目的过程。有关 API 的详细信息，请参阅BatchWrite。

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import {
  BatchWriteCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";
import { readFileSync } from "fs";

// These modules are local to our GitHub repository. We recommend cloning
// the project from GitHub if you want to run this example.
// For more information, see https://github.com/awsdocs/aws-doc-sdk-examples.
import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";
import { chunkArray } from "@aws-doc-sdk-examples/lib/utils/util-array.js";

const dirname = dirnameFromMetaUrl(import.meta.url);

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const file = readFileSync(
    `${dirname}../../../../../resources/sample_files/movies.json`,
  );

  const movies = JSON.parse(file.toString());

  // chunkArray is a local convenience function. It takes an array and returns
  // a generator function. The generator function yields every N items.
  const movieChunks = chunkArray(movies, 25);
```

```
  // For every chunk of 25 movies, make one BatchWrite request.
  for (const chunk of movieChunks) {
    const putRequests = chunk.map((movie) => ({
      PutRequest: {
        Item: movie,
      },
    }));

    const command = new BatchWriteCommand({
      RequestItems: {
        // An existing table is required. A composite key of 'title' and 'year' is
 recommended
        // to account for duplicate titles.
        ["BatchWriteMoviesTable"]: putRequests,
      },
    });

    await docClient.send(command);
  }
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考BatchWriteItem中的。

适用于 JavaScript (v2) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  RequestItems: {
```

```
    TABLE_NAME: [
      {
        PutRequest: {
          Item: {
            KEY: { N: "KEY_VALUE" },
            ATTRIBUTE_1: { S: "ATTRIBUTE_1_VALUE" },
            ATTRIBUTE_2: { N: "ATTRIBUTE_2_VALUE" },
          },
        },
      },
      {
        PutRequest: {
          Item: {
            KEY: { N: "KEY_VALUE" },
            ATTRIBUTE_1: { S: "ATTRIBUTE_1_VALUE" },
            ATTRIBUTE_2: { N: "ATTRIBUTE_2_VALUE" },
          },
        },
      },
    ],
  },
};

ddb.batchWriteItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考BatchWriteItem中的。

场景

开始使用表、项目和查询

以下代码示例展示了如何：

- 创建可保存电影数据的表。

- 在表中加入单一电影，获取并更新此电影。
- 向 JSON 示例文件的表中写入电影数据。
- 查询在给定年份发行的电影。
- 扫描在年份范围内发行的电影。
- 删除表中的电影后再删除表。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { readFileSync } from "fs";
import {
  BillingMode,
  CreateTableCommand,
  DeleteTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";

/**
 * This module is a convenience library. It abstracts Amazon DynamoDB's data type
 * descriptors (such as S, N, B, and BOOL) by marshalling JavaScript objects into
 * AttributeValue shapes.
 */
import {
  BatchWriteCommand,
  DeleteCommand,
  DynamoDBDocumentClient,
  GetCommand,
  PutCommand,
  UpdateCommand,
  paginateQuery,
  paginateScan,
} from "@aws-sdk/lib-dynamodb";

// These modules are local to our GitHub repository. We recommend cloning
```

```
// the project from GitHub if you want to run this example.
// For more information, see https://github.com/awsdocs/aws-doc-sdk-examples.
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";
import { chunkArray } from "@aws-doc-sdk-examples/lib/utils/util-array.js";

const dirname = dirnameFromMetaUrl(import.meta.url);
const tableName = getUniqueName("Movies");
const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

const log = (msg) => console.log(`[SCENARIO] ${msg}`);

export const main = async () => {
  /**
   * Create a table.
   */

  const createTableCommand = new CreateTableCommand({
    TableName: tableName,
    // This example performs a large write to the database.
    // Set the billing mode to PAY_PER_REQUEST to
    // avoid throttling the large write.
    BillingMode: BillingMode.PAY_PER_REQUEST,
    // Define the attributes that are necessary for the key schema.
    AttributeDefinitions: [
      {
        AttributeName: "year",
        // 'N' is a data type descriptor that represents a number type.
        // For a list of all data type descriptors, see the following link.
        // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
        AttributeType: "N",
      },
      { AttributeName: "title", AttributeType: "S" },
    ],
    // The KeySchema defines the primary key. The primary key can be
    // a partition key, or a combination of a partition key and a sort key.
    // Key schema design is important. For more info, see
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/best-
practices.html
    KeySchema: [
      // The way your data is accessed determines how you structure your keys.
      // The movies table will be queried for movies by year. It makes sense
```

```
      // to make year our partition (HASH) key.
      { AttributeName: "year", KeyType: "HASH" },
      { AttributeName: "title", KeyType: "RANGE" },
    ],
  });

  log("Creating a table.");
  const createTableResponse = await client.send(createTableCommand);
  log(`Table created: ${JSON.stringify(createTableResponse.TableDescription)}`);

  // This polls with DescribeTableCommand until the requested table is 'ACTIVE'.
  // You can't write to a table before it's active.
  log("Waiting for the table to be active.");
  await waitUntilTableExists({ client }, { TableName: tableName });
  log("Table active.");

  /**
   * Add a movie to the table.
   */

  log("Adding a single movie to the table.");
  // PutCommand is the first example usage of 'lib-dynamodb'.
  const putCommand = new PutCommand({
    TableName: tableName,
    Item: {
      // In 'client-dynamodb', the AttributeValue would be required (`year: { N:
 1981 }`)
      // 'lib-dynamodb' simplifies the usage ( `year: 1981` )
      year: 1981,
      // The preceding KeySchema defines 'title' as our sort (RANGE) key, so 'title'
      // is required.
      title: "The Evil Dead",
      // Every other attribute is optional.
      info: {
        genres: ["Horror"],
      },
    },
  });
  await docClient.send(putCommand);
  log("The movie was added.");

  /**
   * Get a movie from the table.
   */
```

```
  log("Getting a single movie from the table.");
  const getCommand = new GetCommand({
    TableName: tableName,
    // Requires the complete primary key. For the movies table, the primary key
    // is only the id (partition key).
    Key: {
      year: 1981,
      title: "The Evil Dead",
    },
    // Set this to make sure that recent writes are reflected.
    // For more information, see https://docs.aws.amazon.com/amazondynamodb/latest/
developerguide/HowItWorks.ReadConsistency.html.
    ConsistentRead: true,
  });
  const getResponse = await docClient.send(getCommand);
  log(`Got the movie: ${JSON.stringify(getResponse.Item)}`);

  /**
   * Update a movie in the table.
   */

  log("Updating a single movie in the table.");
  const updateCommand = new UpdateCommand({
    TableName: tableName,
    Key: { year: 1981, title: "The Evil Dead" },
    // This update expression appends "Comedy" to the list of genres.
    // For more information on update expressions, see
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Expressions.UpdateExpressions.html
    UpdateExpression: "set #i.#g = list_append(#i.#g, :vals)",
    ExpressionAttributeNames: { "#i": "info", "#g": "genres" },
    ExpressionAttributeValues: {
      ":vals": ["Comedy"],
    },
    ReturnValues: "ALL_NEW",
  });
  const updateResponse = await docClient.send(updateCommand);
  log(`Movie updated: ${JSON.stringify(updateResponse.Attributes)}`);

  /**
   * Delete a movie from the table.
   */
```

```
  log("Deleting a single movie from the table.");
  const deleteCommand = new DeleteCommand({
    TableName: tableName,
    Key: { year: 1981, title: "The Evil Dead" },
  });
  await client.send(deleteCommand);
  log("Movie deleted.");

  /**
   * Upload a batch of movies.
   */

  log("Adding movies from local JSON file.");
  const file = readFileSync(
    `${dirname}../../../../resources/sample_files/movies.json`,
  );
  const movies = JSON.parse(file.toString());
  // chunkArray is a local convenience function. It takes an array and returns
  // a generator function. The generator function yields every N items.
  const movieChunks = chunkArray(movies, 25);
  // For every chunk of 25 movies, make one BatchWrite request.
  for (const chunk of movieChunks) {
    const putRequests = chunk.map((movie) => ({
      PutRequest: {
        Item: movie,
      },
    }));

    const command = new BatchWriteCommand({
      RequestItems: {
        [tableName]: putRequests,
      },
    });

    await docClient.send(command);
  }
  log("Movies added.");

  /**
   * Query for movies by year.
   */

  log("Querying for all movies from 1981.");
  const paginatedQuery = paginateQuery(
```

```
    { client: docClient },
    {
      TableName: tableName,
      //For more information about query expressions, see
      // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Query.html#Query.KeyConditionExpressions
      KeyConditionExpression: "#y = :y",
      // 'year' is a reserved word in DynamoDB. Indicate that it's an attribute
      // name by using an expression attribute name.
      ExpressionAttributeNames: { "#y": "year" },
      ExpressionAttributeValues: { ":y": 1981 },
      ConsistentRead: true,
    },
  );
  /**
   * @type { Record<string, any>[] };
   */
  const movies1981 = [];
  for await (const page of paginatedQuery) {
    movies1981.push(...page.Items);
  }
  log(`Movies: ${movies1981.map((m) => m.title).join(", ")}`);

  /**
   * Scan the table for movies between 1980 and 1990.
   */

  log(`Scan for movies released between 1980 and 1990`);
  // A 'Scan' operation always reads every item in the table. If your design
 requires
  // the use of 'Scan', consider indexing your table or changing your design.
  // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/bp-query-
scan.html
  const paginatedScan = paginateScan(
    { client: docClient },
    {
      TableName: tableName,
      // Scan uses a filter expression instead of a key condition expression. Scan
 will
      // read the entire table and then apply the filter.
      FilterExpression: "#y between :y1 and :y2",
      ExpressionAttributeNames: { "#y": "year" },
      ExpressionAttributeValues: { ":y1": 1980, ":y2": 1990 },
      ConsistentRead: true,
```

```
    },
  );
  /**
   * @type { Record<string, any>[] };
   */
  const movies1980to1990 = [];
  for await (const page of paginatedScan) {
    movies1980to1990.push(...page.Items);
  }
  log(
    `Movies: ${movies1980to1990
      .map((m) => `${m.title} (${m.year})`)
      .join(", ")}`,
  );

  /**
   * Delete the table.
   */

  const deleteTableCommand = new DeleteTableCommand({ TableName: tableName });
  log(`Deleting table ${tableName}.`);
  await client.send(deleteTableCommand);
  log("Table deleted.");
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考中的以下主题。
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - 查询
  - 扫描
  - UpdateItem

使用批量 PartiQL 语句查询表

以下代码示例展示了如何：

- 通过运行多个 SELECT 语句来获取一批项目。
- 通过运行多个 INSERT 语句来添加一批项目。
- 通过运行多个 UPDATE 语句来更新一批项目。
- 通过运行多个 DELETE 语句来删除一批项目。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

执行批处理 PartiQL 语句。

```
import {
  BillingMode,
  CreateTableCommand,
  DeleteTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";
import {
  DynamoDBDocumentClient,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

const log = (msg) => console.log(`[SCENARIO] ${msg}`);
const tableName = "Cities";

export const main = async () => {
  /**
   * Create a table.
   */
```

```
  log("Creating a table.");
  const createTableCommand = new CreateTableCommand({
    TableName: tableName,
    // This example performs a large write to the database.
    // Set the billing mode to PAY_PER_REQUEST to
    // avoid throttling the large write.
    BillingMode: BillingMode.PAY_PER_REQUEST,
    // Define the attributes that are necessary for the key schema.
    AttributeDefinitions: [
      {
        AttributeName: "name",
        // 'S' is a data type descriptor that represents a number type.
        // For a list of all data type descriptors, see the following link.
        // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
        AttributeType: "S",
      },
    ],
    // The KeySchema defines the primary key. The primary key can be
    // a partition key, or a combination of a partition key and a sort key.
    // Key schema design is important. For more info, see
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/best-
practices.html
    KeySchema: [{ AttributeName: "name", KeyType: "HASH" }],
  });
  await client.send(createTableCommand);
  log(`Table created: ${tableName}.`);

  /**
   * Wait until the table is active.
   */

  // This polls with DescribeTableCommand until the requested table is 'ACTIVE'.
  // You can't write to a table before it's active.
  log("Waiting for the table to be active.");
  await waitUntilTableExists({ client }, { TableName: tableName });
  log("Table active.");

  /**
   * Insert items.
   */

  log("Inserting cities into the table.");
```

```
  const addItemsStatementCommand = new BatchExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.insert.html
    Statements: [
      {
        Statement: `INSERT INTO ${tableName} value {'name':?, 'population':?}`,
        Parameters: ["Alachua", 10712],
      },
      {
        Statement: `INSERT INTO ${tableName} value {'name':?, 'population':?}`,
        Parameters: ["High Springs", 6415],
      },
    ],
  });
  await docClient.send(addItemsStatementCommand);
  log(`Cities inserted.`);

  /**
   * Select items.
   */

  log("Selecting cities from the table.");
  const selectItemsStatementCommand = new BatchExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.select.html
    Statements: [
      {
        Statement: `SELECT * FROM ${tableName} WHERE name=?`,
        Parameters: ["Alachua"],
      },
      {
        Statement: `SELECT * FROM ${tableName} WHERE name=?`,
        Parameters: ["High Springs"],
      },
    ],
  });
  const selectItemResponse = await docClient.send(selectItemsStatementCommand);
  log(
    `Got cities: ${selectItemResponse.Responses.map(
      (r) => `${r.Item.name} (${r.Item.population})`,
    ).join(", ")}`,
  );

  /**
```

```
   * Update items.
   */

  log("Modifying the populations.");
  const updateItemStatementCommand = new BatchExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.update.html
    Statements: [
      {
        Statement: `UPDATE ${tableName} SET population=? WHERE name=?`,
        Parameters: [10, "Alachua"],
      },
      {
        Statement: `UPDATE ${tableName} SET population=? WHERE name=?`,
        Parameters: [5, "High Springs"],
      },
    ],
  });
  await docClient.send(updateItemStatementCommand);
  log(`Updated cities.`);

  /**
   * Delete the items.
   */

  log("Deleting the cities.");
  const deleteItemStatementCommand = new BatchExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.delete.html
    Statements: [
      {
        Statement: `DELETE FROM ${tableName} WHERE name=?`,
        Parameters: ["Alachua"],
      },
      {
        Statement: `DELETE FROM ${tableName} WHERE name=?`,
        Parameters: ["High Springs"],
      },
    ],
  });
  await docClient.send(deleteItemStatementCommand);
  log("Cities deleted.");

  /**
```

```
   * Delete the table.
   */

  log("Deleting the table.");
  const deleteTableCommand = new DeleteTableCommand({ TableName: tableName });
  await client.send(deleteTableCommand);
  log("Table deleted.");
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考BatchExecuteStatement中的。

使用 PartiQL 来查询表

以下代码示例展示了如何：

- 通过运行 SELECT 语句来获取项目。
- 通过运行 INSERT 语句来添加项目。
- 通过运行 UPDATE 语句来更新项目。
- 通过运行 DELETE 语句来删除项目。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

执行单个 PartiQL 语句。

```
import {
  BillingMode,
  CreateTableCommand,
  DeleteTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";
import {
```

```
  DynamoDBDocumentClient,
  ExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

const log = (msg) => console.log(`[SCENARIO] ${msg}`);
const tableName = "SingleOriginCoffees";

export const main = async () => {
  /**
   * Create a table.
   */

  log("Creating a table.");
  const createTableCommand = new CreateTableCommand({
    TableName: tableName,
    // This example performs a large write to the database.
    // Set the billing mode to PAY_PER_REQUEST to
    // avoid throttling the large write.
    BillingMode: BillingMode.PAY_PER_REQUEST,
    // Define the attributes that are necessary for the key schema.
    AttributeDefinitions: [
      {
        AttributeName: "varietal",
        // 'S' is a data type descriptor that represents a number type.
        // For a list of all data type descriptors, see the following link.
        // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
        AttributeType: "S",
      },
    ],
    // The KeySchema defines the primary key. The primary key can be
    // a partition key, or a combination of a partition key and a sort key.
    // Key schema design is important. For more info, see
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/best-
practices.html
    KeySchema: [{ AttributeName: "varietal", KeyType: "HASH" }],
  });
  await client.send(createTableCommand);
  log(`Table created: ${tableName}.`);

  /**
```

```
   * Wait until the table is active.
   */

  // This polls with DescribeTableCommand until the requested table is 'ACTIVE'.
  // You can't write to a table before it's active.
  log("Waiting for the table to be active.");
  await waitUntilTableExists({ client }, { TableName: tableName });
  log("Table active.");

  /**
   * Insert an item.
   */

  log("Inserting a coffee into the table.");
  const addItemStatementCommand = new ExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.insert.html
    Statement: `INSERT INTO ${tableName} value {'varietal':?, 'profile':?}`,
    Parameters: ["arabica", ["chocolate", "floral"]],
  });
  await client.send(addItemStatementCommand);
  log(`Coffee inserted.`);

  /**
   * Select an item.
   */

  log("Selecting the coffee from the table.");
  const selectItemStatementCommand = new ExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.select.html
    Statement: `SELECT * FROM ${tableName} WHERE varietal=?`,
    Parameters: ["arabica"],
  });
  const selectItemResponse = await docClient.send(selectItemStatementCommand);
  log(`Got coffee: ${JSON.stringify(selectItemResponse.Items[0])}`);

  /**
   * Update the item.
   */

  log("Add a flavor profile to the coffee.");
  const updateItemStatementCommand = new ExecuteStatementCommand({
```

```
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.update.html
    Statement: `UPDATE ${tableName} SET profile=list_append(profile, ?) WHERE
 varietal=?`,
    Parameters: [["fruity"], "arabica"],
  });
  await client.send(updateItemStatementCommand);
  log(`Updated coffee`);

  /**
   * Delete the item.
   */

  log("Deleting the coffee.");
  const deleteItemStatementCommand = new ExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.delete.html
    Statement: `DELETE FROM ${tableName} WHERE varietal=?`,
    Parameters: ["arabica"],
  });
  await docClient.send(deleteItemStatementCommand);
  log("Coffee deleted.");

  /**
   * Delete the table.
   */

  log("Deleting the table.");
  const deleteTableCommand = new DeleteTableCommand({ TableName: tableName });
  await client.send(deleteTableCommand);
  log("Table deleted.");
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考ExecuteStatement中的。

使用适用于 JavaScript (v3) 的软件开发工具包的 Amazon EC2 示例

以下代码示例向您展示了如何在 Amazon EC2 中使用 Amazon SDK for JavaScript (v3) 来执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景 是展示如何通过在同一服务中调用多个函数来完成特定任务的代码示例。

每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关如何在上下文中设置和运行代码的说明。

开始使用

Hello Amazon EC2

以下代码示例展示了如何开始使用 Amazon EC2。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { DescribeSecurityGroupsCommand } from "@aws-sdk/client-ec2";

import { client } from "./libs/client.js";

// Call DescribeSecurityGroups and display the result.
export const main = async () => {
  try {
    const { SecurityGroups } = await client.send(
      new DescribeSecurityGroupsCommand({}),
    );

    const securityGroupList = SecurityGroups.slice(0, 9)
      .map((sg) => ` • ${sg.GroupId}: ${sg.GroupName}`)
      .join("\n");

    console.log(
      "Hello, Amazon EC2! Let's list up to 10 of your security groups:",
    );
    console.log(securityGroupList);
  } catch (err) {
```

```
    console.error(err);
  }
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DescribeSecurityGroups中的。

主题

- 操作
- 场景

操作

分配弹性 IP 地址

以下代码示例展示了如何为 Amazon EC2 分配弹性 IP 地址。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { AllocateAddressCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

export const main = async () => {
  const command = new AllocateAddressCommand({});

  try {
    const { AllocationId, PublicIp } = await client.send(command);
    console.log("A new IP address has been allocated to your account:");
    console.log(`ID: ${AllocationId} Public IP: ${PublicIp}`);
    console.log(
      "You can view your IP addresses in the AWS Management Console for Amazon EC2.
 Look under Network & Security > Elastic IPs",
```

```
    );
  } catch (err) {
    console.error(err);
  }
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考AllocateAddress中的。

关联弹性 IP 地址和实例

以下代码示例展示了如何将弹性 IP 地址与 Amazon EC2 实例关联。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { AssociateAddressCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

export const main = async () => {
  // You need to allocate an Elastic IP address before associating it with an
 instance.
  // You can do that with the AllocateAddressCommand.
  const allocationId = "ALLOCATION_ID";
  // You need to create an EC2 instance before an IP address can be associated with
 it.
  // You can do that with the RunInstancesCommand.
  const instanceId = "INSTANCE_ID";
  const command = new AssociateAddressCommand({
    AllocationId: allocationId,
    InstanceId: instanceId,
  });

  try {
    const { AssociationId } = await client.send(command);
```

```
    console.log(
      `Address with allocation ID ${allocationId} is now associated with instance
 ${instanceId}.`,
      `The association ID is ${AssociationId}.`,
    );
  } catch (err) {
    console.error(err);
  }
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考AssociateAddress中的。

创建启动模板

以下代码示例演示了如何创建 Amazon EC2 启动模板。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    const ssmClient = new SSMClient({});
    const { Parameter } = await ssmClient.send(
      new GetParameterCommand({
        Name: "/aws/service/ami-amazon-linux-latest/amzn2-ami-hvm-x86_64-gp2",
      }),
    );
    const ec2Client = new EC2Client({});
    await ec2Client.send(
      new CreateLaunchTemplateCommand({
        LaunchTemplateName: NAMES.launchTemplateName,
        LaunchTemplateData: {
          InstanceType: "t3.micro",
          ImageId: Parameter.Value,
          IamInstanceProfile: { Name: NAMES.instanceProfileName },
          UserData: readFileSync(
            join(RESOURCES_PATH, "server_startup_script.sh"),
```

```
          ).toString("base64"),
          KeyName: NAMES.keyPairName,
        },
      }),
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考CreateLaunchTemplate中的。

创建安全组

以下代码示例展示了如何创建 Amazon EC2 安全组。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { CreateSecurityGroupCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

export const main = async () => {
  const command = new CreateSecurityGroupCommand({
    // Up to 255 characters in length. Cannot start with sg-.
    GroupName: "SECURITY_GROUP_NAME",
    // Up to 255 characters in length.
    Description: "DESCRIPTION",
  });

  try {
    const { GroupId } = await client.send(command);
    console.log(GroupId);
  } catch (err) {
    console.error(err);
  }
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考CreateSecurityGroup中的。

创建安全密钥对

以下代码示例展示了如何为 Amazon EC2 创建安全密钥对。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { CreateKeyPairCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

export const main = async () => {
  try {
    // Create a key pair in Amazon EC2.
    const { KeyMaterial, KeyName } = await client.send(
      // A unique name for the key pair. Up to 255 ASCII characters.
      new CreateKeyPairCommand({ KeyName: "KEY_PAIR_NAME" }),
    );
    // This logs your private key. Be sure to save it.
    console.log(KeyName);
    console.log(KeyMaterial);
  } catch (err) {
    console.error(err);
  }
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考CreateKeyPair中的。

创建并运行实例

以下代码示例展示了如何创建并运行 Amazon EC2 实例。

适用于 JavaScript (v3) 的软件开发工具包

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { RunInstancesCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

// Create a new EC2 instance.
export const main = async () => {
  const command = new RunInstancesCommand({
    // Your key pair name.
    KeyName: "KEY_PAIR_NAME",
    // Your security group.
    SecurityGroupIds: ["SECURITY_GROUP_ID"],
    // An x86_64 compatible image.
    ImageId: "ami-0001a0d1a04bfcc30",
    // An x86_64 compatible free-tier instance type.
    InstanceType: "t1.micro",
    // Ensure only 1 instance launches.
    MinCount: 1,
    MaxCount: 1,
  });

  try {
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考RunInstances中的。

删除启动模板

以下代码示例演示了如何删除 Amazon EC2 启动模板。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
      await client.send(
        new DeleteLaunchTemplateCommand({
          LaunchTemplateName: NAMES.launchTemplateName,
        }),
      );
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DeleteLaunchTemplate中的。

删除安全组

以下代码示例展示了如何删除 Amazon EC2 安全组。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { DeleteSecurityGroupCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

export const main = async () => {
  const command = new DeleteSecurityGroupCommand({
    GroupId: "GROUP_ID",
  });

  try {
```

```
    await client.send(command);
    console.log("Security group deleted successfully.");
  } catch (err) {
    console.error(err);
  }
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DeleteSecurityGroup中的。

删除安全密钥对

以下代码示例展示了如何删除 Amazon EC2 安全密钥对。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { DeleteKeyPairCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

export const main = async () => {
  const command = new DeleteKeyPairCommand({
    KeyName: "KEY_PAIR_NAME",
  });

  try {
    await client.send(command);
    console.log("Successfully deleted key pair.");
  } catch (err) {
    console.error(err);
  }
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DeleteKeyPair中的。

描述区域

以下代码示例展示了如何描述 Amazon EC2 区域。

适用于 JavaScript (v3) 的软件开发工具包

Note

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { DescribeRegionsCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

export const main = async () => {
  const command = new DescribeRegionsCommand({
    // By default this command will not show regions that require you to opt-in.
    // When AllRegions true even the regions that require opt-in will be returned.
    AllRegions: true,
    // You can omit the Filters property if you want to get all regions.
    Filters: [
      {
        Name: "region-name",
        // You can specify multiple values for a filter.
        // You can also use '*' as a wildcard. This will return all
        // of the regions that start with `us-east-`.
        Values: ["ap-southeast-4"],
      },
    ],
  });

  try {
    const { Regions } = await client.send(command);
    const regionsList = Regions.map((reg) => ` • ${reg.RegionName}`);
    console.log("Found regions:");
    console.log(regionsList.join("\n"));
  } catch (err) {
    console.error(err);
  }
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考[DescribeRegions](DescribeRegions)中的。

描述实例

以下代码示例展示了如何描述 Amazon EC2 实例。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { DescribeInstancesCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

// List all of your EC2 instances running with x86_64 architecture that were
// launched this month.
export const main = async () => {
  const d = new Date();
  const year = d.getFullYear();
  const month = `0${d.getMonth() + 1}`.slice(-2);
  const launchTimePattern = `${year}-${month}-*`;
  const command = new DescribeInstancesCommand({
    Filters: [
      { Name: "architecture", Values: ["x86_64"] },
      { Name: "instance-state-name", Values: ["running"] },
      {
        Name: "launch-time",
        Values: [launchTimePattern],
      },
    ],
  });

  try {
    const { Reservations } = await client.send(command);
    const instanceList = Reservations.reduce((prev, current) => {
```

```
      return prev.concat(current.Instances);
    }, []);

    console.log(instanceList);
  } catch (err) {
    console.error(err);
  }
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DescribeInstances中的。

禁用详细监控

以下代码示例展示了如何禁用 Amazon EC2 实例的详细监控。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { UnmonitorInstancesCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

export const main = async () => {
  const command = new UnmonitorInstancesCommand({
    InstanceIds: ["i-09a3dfe7ae00e853f"],
  });

  try {
    const { InstanceMonitorings } = await client.send(command);
    const instanceMonitoringsList = InstanceMonitorings.map(
      (im) =>
        ` • Detailed monitoring state for ${im.InstanceId} is
 ${im.Monitoring.State}.`,
    );
    console.log("Monitoring status:");
```

```
    console.log(instanceMonitoringsList.join("\n"));
  } catch (err) {
    console.error(err);
  }
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考UnmonitorInstances中的。

取消弹性 IP 地址与实例的关联

以下代码示例展示了如何取消弹性 IP 地址与 Amazon EC2 实例的关联。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { DisassociateAddressCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

// Disassociate an Elastic IP address from an instance.
export const main = async () => {
  const command = new DisassociateAddressCommand({
    // You can also use PublicIp, but that is for EC2 classic which is being
 retired.
    AssociationId: "ASSOCIATION_ID",
  });

  try {
    await client.send(command);
    console.log("Successfully disassociated address");
  } catch (err) {
    console.error(err);
  }
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DisassociateAddress中的。

启用监控

以下代码示例展示了如何为正在运行的 Amazon EC2 实例启用监控。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { MonitorInstancesCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

// Turn on detailed monitoring for the selected instance.
// By default, metrics are sent to Amazon CloudWatch every 5 minutes.
// For a cost you can enable detailed monitoring which sends metrics every minute.
export const main = async () => {
  const command = new MonitorInstancesCommand({
    InstanceIds: ["INSTANCE_ID"],
  });

  try {
    const { InstanceMonitorings } = await client.send(command);
    const instancesBeingMonitored = InstanceMonitorings.map(
      (im) =>
        ` • Detailed monitoring state for ${im.InstanceId} is
 ${im.Monitoring.State}.`,
    );
    console.log("Monitoring status:");
    console.log(instancesBeingMonitored.join("\n"));
  } catch (err) {
    console.error(err);
  }
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考MonitorInstances中的。

获取有关 Amazon 机器映像的数据

以下代码示例展示了如何获取有关 Amazon 机器映像 (AMI) 的数据。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { paginateDescribeImages } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

// List at least the first i386 image available for EC2 instances.
export const main = async () => {
  // The paginate function is a wrapper around the base command.
  const paginator = paginateDescribeImages(
    // Without limiting the page size, this call can take a long time. pageSize is
 just sugar for
    // the MaxResults property in the base command.
    { client, pageSize: 25 },
    {
      // There are almost 70,000 images available. Be specific with your filtering
      // to increase efficiency.
      // See https://docs.aws.amazon.com/AWSJavaScriptSDK/v3/latest/clients/client-
ec2/interfaces/describeimagescommandinput.html#filters
      Filters: [{ Name: "architecture", Values: ["x86_64"] }],
    },
  );

  try {
    const arm64Images = [];
    for await (const page of paginator) {
      if (page.Images.length) {
        arm64Images.push(...page.Images);
        // Once we have at least 1 result, we can stop.
        if (arm64Images.length >= 1) {
          break;
        }
      }
```

```
    }
    console.log(arm64Images);
  } catch (err) {
    console.error(err);
  }
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DescribeImages中的。

获取有关安全组的数据

以下代码示例展示了如何获取有关 Amazon EC2 安全组的数据。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { DescribeSecurityGroupsCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

// Log the details of a specific security group.
export const main = async () => {
  const command = new DescribeSecurityGroupsCommand({
    GroupIds: ["SECURITY_GROUP_ID"],
  });

  try {
    const { SecurityGroups } = await client.send(command);
    console.log(JSON.stringify(SecurityGroups, null, 2));
  } catch (err) {
    console.error(err);
  }
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DescribeSecurityGroups中的。

获取有关实例类型的数据

以下代码示例展示了如何获取有关 Amazon EC2 实例类型的数据。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import {
  paginateDescribeInstanceTypes,
  DescribeInstanceTypesCommand,
} from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

// List at least the first arm64 EC2 instance type available.
export const main = async () => {
  // The paginate function is a wrapper around the underlying command.
  const paginator = paginateDescribeInstanceTypes(
    // Without limiting the page size, this call can take a long time. pageSize is
 just sugar for
    // the MaxResults property in the underlying command.
    { client, pageSize: 25 },
    {
      Filters: [
        { Name: "processor-info.supported-architecture", Values: ["x86_64"] },
        { Name: "free-tier-eligible", Values: ["true"] },
      ],
    }
  );

  try {
    const instanceTypes = [];

    for await (const page of paginator) {
```

```
      if (page.InstanceTypes.length) {
        instanceTypes.push(...page.InstanceTypes);

        // When we have at least 1 result, we can stop.
        if (instanceTypes.length >= 1) {
          break;
        }
      }
    }
    console.log(instanceTypes);
  } catch (err) {
    console.error(err);
  }
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DescribeInstanceTypes中的。

获取有关与实例关联的实例配置文件的数据

以下代码示例演示了如何获取有关与 Amazon EC2 实例关联的实例配置文件的数据。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
      const ec2Client = new EC2Client({});
      const { IamInstanceProfileAssociations } = await ec2Client.send(
        new DescribeIamInstanceProfileAssociationsCommand({
          Filters: [
            { Name: "instance-id", Values: [state.targetInstance.InstanceId] },
          ],
        }),
      );
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DescribeIamInstanceProfileAssociations中的。

获取有关弹性 IP 地址的详细信息

以下代码示例展示了如何获取有关弹性 IP 地址的详细信息。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { DescribeAddressesCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

export const main = async () => {
  const command = new DescribeAddressesCommand({
    // You can omit this property to show all addresses.
    AllocationIds: ["ALLOCATION_ID"],
  });

  try {
    const { Addresses } = await client.send(command);
    const addressList = Addresses.map((address) => ` • ${address.PublicIp}`);
    console.log("Elastic IP addresses:");
    console.log(addressList.join("\n"));
  } catch (err) {
    console.error(err);
  }
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DescribeAddresses中的。

获取默认 VPC

以下代码示例演示了如何获取当前账户的默认 VPC。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
const client = new EC2Client({});
const { Vpcs } = await client.send(
  new DescribeVpcsCommand({
    Filters: [{ Name: "is-default", Values: ["true"] }],
  }),
);
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DescribeVpcs中的。

获取 VPC 的默认子网

以下代码示例演示了如何获取 VPC 的默认子网。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
const client = new EC2Client({});
const { Subnets } = await client.send(
  new DescribeSubnetsCommand({
    Filters: [
      { Name: "vpc-id", Values: [state.defaultVpc] },
      { Name: "availability-zone", Values: state.availabilityZoneNames },
      { Name: "default-for-az", Values: ["true"] },
    ],
  }),
);
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DescribeSubnets中的。

列出安全密钥对

以下代码示例展示了如何列出 Amazon EC2 安全密钥对。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { DescribeKeyPairsCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

export const main = async () => {
  const command = new DescribeKeyPairsCommand({});

  try {
    const { KeyPairs } = await client.send(command);
    const keyPairList = KeyPairs.map(
      (kp) => ` • ${kp.KeyPairId}: ${kp.KeyName}`,
    ).join("\n");
    console.log("The following key pairs were found in your account:");
    console.log(keyPairList);
  } catch (err) {
    console.error(err);
  }
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DescribeKeyPairs中的。

重启实例

以下代码示例展示了如何重启 Amazon EC2 实例。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { RebootInstancesCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

export const main = async () => {
  const command = new RebootInstancesCommand({
    InstanceIds: ["INSTANCE_ID"],
  });

  try {
    await client.send(command);
    console.log("Instance rebooted successfully.");
  } catch (err) {
    console.error(err);
  }
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考RebootInstances中的。

释放弹性 IP 地址

以下代码示例展示了如何释放弹性 IP 地址。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { ReleaseAddressCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

export const main = async () => {
  const command = new ReleaseAddressCommand({
    // You can also use PublicIp, but that is for EC2 classic which is being
 retired.
    AllocationId: "ALLOCATION_ID",
  });

  try {
    await client.send(command);
    console.log("Successfully released address.");
  } catch (err) {
    console.error(err);
  }
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考ReleaseAddress中的。

替换与实例关联的实例配置文件

以下代码示例演示了如何替换与 Amazon EC2 实例关联的实例配置文件。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
      await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
        ec2Client.send(
          new ReplaceIamInstanceProfileAssociationCommand({
            AssociationId: state.instanceProfileAssociationId,
            IamInstanceProfile: { Name: NAMES.ssmOnlyInstanceProfileName },
          }),
        ),
```

```
);
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考[ReplaceIamInstanceProfileAssociation](ReplaceIamInstanceProfileAssociation)中的。

为安全组设置入站规则

以下代码示例展示了如何为 Amazon EC2 安全组设置入站规则。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { AuthorizeSecurityGroupIngressCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

// Grant permissions for a single IP address to ssh into instances
// within the provided security group.
export const main = async () => {
  const command = new AuthorizeSecurityGroupIngressCommand({
    // Replace with a security group ID from the AWS console or
    // the DescribeSecurityGroupsCommand.
    GroupId: "SECURITY_GROUP_ID",
    IpPermissions: [
      {
        IpProtocol: "tcp",
        FromPort: 22,
        ToPort: 22,
        // Replace 0.0.0.0 with the IP address to authorize.
        // For more information on this notation, see
        // https://en.wikipedia.org/wiki/Classless_Inter-
Domain_Routing#CIDR_notation
        IpRanges: [{ CidrIp: "0.0.0.0/32" }],
      },
    ],
```

```
  });

  try {
    const { SecurityGroupRules } = await client.send(command);
    console.log(JSON.stringify(SecurityGroupRules, null, 2));
  } catch (err) {
    console.error(err);
  }
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考AuthorizeSecurityGroupIngress中的。

启动实例

以下代码示例展示了如何启动 Amazon EC2 实例。

适用于 JavaScript (v3) 的软件开发工具包

> Note
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { StartInstancesCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

export const main = async () => {
  const command = new StartInstancesCommand({
    // Use DescribeInstancesCommand to find InstanceIds
    InstanceIds: ["INSTANCE_ID"],
  });

  try {
    const { StartingInstances } = await client.send(command);
    const instanceIdList = StartingInstances.map(
      (instance) => ` • ${instance.InstanceId}`,
    );
```

```
    console.log("Starting instances:");
    console.log(instanceIdList.join("\n"));
  } catch (err) {
    console.error(err);
  }
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考StartInstances中的。

停止实例

以下代码示例展示了如何停止 Amazon EC2 实例。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { StopInstancesCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

export const main = async () => {
  const command = new StopInstancesCommand({
    // Use DescribeInstancesCommand to find InstanceIds
    InstanceIds: ["INSTANCE_ID"],
  });

  try {
    const { StoppingInstances } = await client.send(command);
    const instanceIdList = StoppingInstances.map(
      (instance) => ` • ${instance.InstanceId}`,
    );
    console.log("Stopping instances:");
    console.log(instanceIdList.join("\n"));
  } catch (err) {
    console.error(err);
```

```
  }
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考StopInstances中的。

终止实例

以下代码示例展示了如何终止 Amazon EC2 实例。

适用于 JavaScript (v3) 的软件开发工具包

> Note
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { TerminateInstancesCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

export const main = async () => {
  const command = new TerminateInstancesCommand({
    InstanceIds: ["INSTANCE_ID"],
  });

  try {
    const { TerminatingInstances } = await client.send(command);
    const instanceList = TerminatingInstances.map(
      (instance) => ` • ${instance.InstanceId}`,
    );
    console.log("Terminating instances:");
    console.log(instanceList.join("\n"));
  } catch (err) {
    console.error(err);
  }
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考TerminateInstances中的。

场景

构建和管理弹性服务

以下代码示例演示了如何创建可返回书籍、电影和歌曲推荐的负载均衡的 Web 服务。该示例演示服务如何响应故障，以及如何重组服务以提高故障发生时的弹性。

- 使用 Amazon EC2 Auto Scaling 组根据启动模板创建 Amazon Elastic Compute Cloud（Amazon EC2）实例，并将实例数量保持在指定范围内。
- 使用弹性负载均衡处理和分发 HTTP 请求。
- 监控自动扩缩组中实例的运行状况，并仅将请求转发到运行状况良好的实例。
- 在每个 EC2 实例上运行 Python Web 服务器以处理 HTTP 请求。Web 服务器以建议和运行状况检查作为响应。
- 使用 Amazon DynamoDB 表模拟推荐服务。
- 通过更新 Amazon Systems Manager 参数来控制 Web 服务器对请求和运行状况检查的响应。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 [Amazon 代码示例存储库](#)中查找完整示例，了解如何进行设置和运行。

在命令提示符中运行交互式场景。

```
#!/usr/bin/env node
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0


import {
  Scenario,
  parseScenarioArgs,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

/**
 * The workflow steps are split into three stages:
 *   - deploy
```

```
 *   - demo
 *   - destroy
 *
 * Each of these stages has a corresponding file prefixed with steps-*.
 */
import { deploySteps } from "./steps-deploy.js";
import { demoSteps } from "./steps-demo.js";
import { destroySteps } from "./steps-destroy.js";

/**
 * The context is passed to every scenario. Scenario steps
 * will modify the context.
 */
const context = {};

/**
 * Three Scenarios are created for the workflow. A Scenario is an orchestration
 class
 * that simplifies running a series of steps.
 */
export const scenarios = {
  // Deploys all resources necessary for the workflow.
  deploy: new Scenario("Resilient Workflow - Deploy", deploySteps, context),
  // Demonstrates how a fragile web service can be made more resilient.
  demo: new Scenario("Resilient Workflow - Demo", demoSteps, context),
  // Destroys the resources created for the workflow.
  destroy: new Scenario("Resilient Workflow - Destroy", destroySteps, context),
};

// Call function if run directly
import { fileURLToPath } from "url";

if (process.argv[1] === fileURLToPath(import.meta.url)) {
  parseScenarioArgs(scenarios);
}
```

创建部署所有资源的步骤。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { join } from "node:path";
import { readFileSync, writeFileSync } from "node:fs";
```

```
import axios from "axios";

import {
  BatchWriteItemCommand,
  CreateTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";
import {
  EC2Client,
  CreateKeyPairCommand,
  CreateLaunchTemplateCommand,
  DescribeAvailabilityZonesCommand,
  DescribeVpcsCommand,
  DescribeSubnetsCommand,
  DescribeSecurityGroupsCommand,
  AuthorizeSecurityGroupIngressCommand,
} from "@aws-sdk/client-ec2";
import {
  IAMClient,
  CreatePolicyCommand,
  CreateRoleCommand,
  CreateInstanceProfileCommand,
  AddRoleToInstanceProfileCommand,
  AttachRolePolicyCommand,
  waitUntilInstanceProfileExists,
} from "@aws-sdk/client-iam";
import { SSMClient, GetParameterCommand } from "@aws-sdk/client-ssm";
import {
  CreateAutoScalingGroupCommand,
  AutoScalingClient,
  AttachLoadBalancerTargetGroupsCommand,
} from "@aws-sdk/client-auto-scaling";
import {
  CreateListenerCommand,
  CreateLoadBalancerCommand,
  CreateTargetGroupCommand,
  ElasticLoadBalancingV2Client,
  waitUntilLoadBalancerAvailable,
} from "@aws-sdk/client-elastic-load-balancing-v2";

import {
  ScenarioOutput,
  ScenarioInput,
```

```
  ScenarioAction,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES, RESOURCES_PATH, ROOT } from "./constants.js";
import { initParamsSteps } from "./steps-reset-params.js";

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const deploySteps = [
  new ScenarioOutput("introduction", MESSAGES.introduction, { header: true }),
  new ScenarioInput("confirmDeployment", MESSAGES.confirmDeployment, {
    type: "confirm",
  }),
  new ScenarioAction(
    "handleConfirmDeployment",
    (c) => c.confirmDeployment === false && process.exit(),
  ),
  new ScenarioOutput(
    "creatingTable",
    MESSAGES.creatingTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
  new ScenarioAction("createTable", async () => {
    const client = new DynamoDBClient({});
    await client.send(
      new CreateTableCommand({
        TableName: NAMES.tableName,
        ProvisionedThroughput: {
          ReadCapacityUnits: 5,
          WriteCapacityUnits: 5,
        },
        AttributeDefinitions: [
          {
            AttributeName: "MediaType",
            AttributeType: "S",
          },
          {
            AttributeName: "ItemId",
            AttributeType: "N",
          },
        ],
        KeySchema: [
          {
```

```
            AttributeName: "MediaType",
            KeyType: "HASH",
          },
          {
            AttributeName: "ItemId",
            KeyType: "RANGE",
          },
        ],
      }),
    );
    await waitUntilTableExists({ client }, { TableName: NAMES.tableName });
  }),
  new ScenarioOutput(
    "createdTable",
    MESSAGES.createdTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
  new ScenarioOutput(
    "populatingTable",
    MESSAGES.populatingTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
  new ScenarioAction("populateTable", () => {
    const client = new DynamoDBClient({});
    /**
     * @type {{ default: import("@aws-sdk/client-dynamodb").PutRequest['Item'][] }}
     */
    const recommendations = JSON.parse(
      readFileSync(join(RESOURCES_PATH, "recommendations.json")),
    );

    return client.send(
      new BatchWriteItemCommand({
        RequestItems: {
          [NAMES.tableName]: recommendations.map((item) => ({
            PutRequest: { Item: item },
          })),
        },
      }),
    );
  }),
  new ScenarioOutput(
    "populatedTable",
    MESSAGES.populatedTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
  new ScenarioOutput(
```

```
    "creatingKeyPair",
    MESSAGES.creatingKeyPair.replace("${KEY_PAIR_NAME}", NAMES.keyPairName),
  ),
  new ScenarioAction("createKeyPair", async () => {
    const client = new EC2Client({});
    const { KeyMaterial } = await client.send(
      new CreateKeyPairCommand({
        KeyName: NAMES.keyPairName,
      }),
    );

    writeFileSync(`${NAMES.keyPairName}.pem`, KeyMaterial, { mode: 0o600 });
  }),
  new ScenarioOutput(
    "createdKeyPair",
    MESSAGES.createdKeyPair.replace("${KEY_PAIR_NAME}", NAMES.keyPairName),
  ),
  new ScenarioOutput(
    "creatingInstancePolicy",
    MESSAGES.creatingInstancePolicy.replace(
      "${INSTANCE_POLICY_NAME}",
      NAMES.instancePolicyName,
    ),
  ),
  new ScenarioAction("createInstancePolicy", async (state) => {
    const client = new IAMClient({});
    const {
      Policy: { Arn },
    } = await client.send(
      new CreatePolicyCommand({
        PolicyName: NAMES.instancePolicyName,
        PolicyDocument: readFileSync(
          join(RESOURCES_PATH, "instance_policy.json"),
        ),
      }),
    );
    state.instancePolicyArn = Arn;
  }),
  new ScenarioOutput("createdInstancePolicy", (state) =>
    MESSAGES.createdInstancePolicy
      .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
      .replace("${INSTANCE_POLICY_ARN}", state.instancePolicyArn),
  ),
  new ScenarioOutput(
```

```
      "creatingInstanceRole",
      MESSAGES.creatingInstanceRole.replace(
        "${INSTANCE_ROLE_NAME}",
        NAMES.instanceRoleName,
      ),
    ),
    new ScenarioAction("createInstanceRole", () => {
      const client = new IAMClient({});
      return client.send(
        new CreateRoleCommand({
          RoleName: NAMES.instanceRoleName,
          AssumeRolePolicyDocument: readFileSync(
            join(ROOT, "assume-role-policy.json"),
          ),
        }),
      );
    }),
    new ScenarioOutput(
      "createdInstanceRole",
      MESSAGES.createdInstanceRole.replace(
        "${INSTANCE_ROLE_NAME}",
        NAMES.instanceRoleName,
      ),
    ),
    new ScenarioOutput(
      "attachingPolicyToRole",
      MESSAGES.attachingPolicyToRole
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName)
        .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName),
    ),
    new ScenarioAction("attachPolicyToRole", async (state) => {
      const client = new IAMClient({});
      await client.send(
        new AttachRolePolicyCommand({
          RoleName: NAMES.instanceRoleName,
          PolicyArn: state.instancePolicyArn,
        }),
      );
    }),
    new ScenarioOutput(
      "attachedPolicyToRole",
      MESSAGES.attachedPolicyToRole
        .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
```

```
  ),
  new ScenarioOutput(
    "creatingInstanceProfile",
    MESSAGES.creatingInstanceProfile.replace(
      "${INSTANCE_PROFILE_NAME}",
      NAMES.instanceProfileName,
    ),
  ),
  new ScenarioAction("createInstanceProfile", async (state) => {
    const client = new IAMClient({});
    const {
      InstanceProfile: { Arn },
    } = await client.send(
      new CreateInstanceProfileCommand({
        InstanceProfileName: NAMES.instanceProfileName,
      }),
    );
    state.instanceProfileArn = Arn;

    await waitUntilInstanceProfileExists(
      { client },
      { InstanceProfileName: NAMES.instanceProfileName },
    );
  }),
  new ScenarioOutput("createdInstanceProfile", (state) =>
    MESSAGES.createdInstanceProfile
      .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
      .replace("${INSTANCE_PROFILE_ARN}", state.instanceProfileArn),
  ),
  new ScenarioOutput(
    "addingRoleToInstanceProfile",
    MESSAGES.addingRoleToInstanceProfile
      .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
  ),
  new ScenarioAction("addRoleToInstanceProfile", () => {
    const client = new IAMClient({});
    return client.send(
      new AddRoleToInstanceProfileCommand({
        RoleName: NAMES.instanceRoleName,
        InstanceProfileName: NAMES.instanceProfileName,
      }),
    );
  }),
```

```
  new ScenarioOutput(
    "addedRoleToInstanceProfile",
    MESSAGES.addedRoleToInstanceProfile
      .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
  ),
  ...initParamsSteps,
  new ScenarioOutput("creatingLaunchTemplate", MESSAGES.creatingLaunchTemplate),
  new ScenarioAction("createLaunchTemplate", async () => {
    // snippet-start:[javascript.v3.wkflw.resilient.CreateLaunchTemplate]
    const ssmClient = new SSMClient({});
    const { Parameter } = await ssmClient.send(
      new GetParameterCommand({
        Name: "/aws/service/ami-amazon-linux-latest/amzn2-ami-hvm-x86_64-gp2",
      }),
    );
    const ec2Client = new EC2Client({});
    await ec2Client.send(
      new CreateLaunchTemplateCommand({
        LaunchTemplateName: NAMES.launchTemplateName,
        LaunchTemplateData: {
          InstanceType: "t3.micro",
          ImageId: Parameter.Value,
          IamInstanceProfile: { Name: NAMES.instanceProfileName },
          UserData: readFileSync(
            join(RESOURCES_PATH, "server_startup_script.sh"),
          ).toString("base64"),
          KeyName: NAMES.keyPairName,
        },
      }),
      // snippet-end:[javascript.v3.wkflw.resilient.CreateLaunchTemplate]
    );
  }),
  new ScenarioOutput(
    "createdLaunchTemplate",
    MESSAGES.createdLaunchTemplate.replace(
      "${LAUNCH_TEMPLATE_NAME}",
      NAMES.launchTemplateName,
    ),
  ),
  new ScenarioOutput(
    "creatingAutoScalingGroup",
    MESSAGES.creatingAutoScalingGroup.replace(
      "${AUTO_SCALING_GROUP_NAME}",
```

```
      NAMES.autoScalingGroupName,
    ),
  ),
  new ScenarioAction("createAutoScalingGroup", async (state) => {
    const ec2Client = new EC2Client({});
    const { AvailabilityZones } = await ec2Client.send(
      new DescribeAvailabilityZonesCommand({}),
    );
    state.availabilityZoneNames = AvailabilityZones.map((az) => az.ZoneName);
    const autoScalingClient = new AutoScalingClient({});
    await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
      autoScalingClient.send(
        new CreateAutoScalingGroupCommand({
          AvailabilityZones: state.availabilityZoneNames,
          AutoScalingGroupName: NAMES.autoScalingGroupName,
          LaunchTemplate: {
            LaunchTemplateName: NAMES.launchTemplateName,
            Version: "$Default",
          },
          MinSize: 3,
          MaxSize: 3,
        }),
      ),
    );
  }),
  new ScenarioOutput(
    "createdAutoScalingGroup",
    /**
     * @param {{ availabilityZoneNames: string[] }} state
     */
    (state) =>
      MESSAGES.createdAutoScalingGroup
        .replace("${AUTO_SCALING_GROUP_NAME}", NAMES.autoScalingGroupName)
        .replace(
          "${AVAILABILITY_ZONE_NAMES}",
          state.availabilityZoneNames.join(", "),
        ),
  ),
  new ScenarioInput("confirmContinue", MESSAGES.confirmContinue, {
    type: "confirm",
  }),
  new ScenarioOutput("loadBalancer", MESSAGES.loadBalancer),
  new ScenarioOutput("gettingVpc", MESSAGES.gettingVpc),
  new ScenarioAction("getVpc", async (state) => {
```

```
    // snippet-start:[javascript.v3.wkflw.resilient.DescribeVpcs]
    const client = new EC2Client({});
    const { Vpcs } = await client.send(
      new DescribeVpcsCommand({
        Filters: [{ Name: "is-default", Values: ["true"] }],
      }),
    );
    // snippet-end:[javascript.v3.wkflw.resilient.DescribeVpcs]
    state.defaultVpc = Vpcs[0].VpcId;
  }),
  new ScenarioOutput("gotVpc", (state) =>
    MESSAGES.gotVpc.replace("${VPC_ID}", state.defaultVpc),
  ),
  new ScenarioOutput("gettingSubnets", MESSAGES.gettingSubnets),
  new ScenarioAction("getSubnets", async (state) => {
    // snippet-start:[javascript.v3.wkflw.resilient.DescribeSubnets]
    const client = new EC2Client({});
    const { Subnets } = await client.send(
      new DescribeSubnetsCommand({
        Filters: [
          { Name: "vpc-id", Values: [state.defaultVpc] },
          { Name: "availability-zone", Values: state.availabilityZoneNames },
          { Name: "default-for-az", Values: ["true"] },
        ],
      }),
    );
    // snippet-end:[javascript.v3.wkflw.resilient.DescribeSubnets]
    state.subnets = Subnets.map((subnet) => subnet.SubnetId);
  }),
  new ScenarioOutput(
    "gotSubnets",
    /**
     * @param {{ subnets: string[] }} state
     */
    (state) =>
      MESSAGES.gotSubnets.replace("${SUBNETS}", state.subnets.join(", ")),
  ),
  new ScenarioOutput(
    "creatingLoadBalancerTargetGroup",
    MESSAGES.creatingLoadBalancerTargetGroup.replace(
      "${TARGET_GROUP_NAME}",
      NAMES.loadBalancerTargetGroupName,
    ),
  ),
```

```
  new ScenarioAction("createLoadBalancerTargetGroup", async (state) => {
    // snippet-start:[javascript.v3.wkflw.resilient.CreateTargetGroup]
    const client = new ElasticLoadBalancingV2Client({});
    const { TargetGroups } = await client.send(
      new CreateTargetGroupCommand({
        Name: NAMES.loadBalancerTargetGroupName,
        Protocol: "HTTP",
        Port: 80,
        HealthCheckPath: "/healthcheck",
        HealthCheckIntervalSeconds: 10,
        HealthCheckTimeoutSeconds: 5,
        HealthyThresholdCount: 2,
        UnhealthyThresholdCount: 2,
        VpcId: state.defaultVpc,
      }),
    );
    // snippet-end:[javascript.v3.wkflw.resilient.CreateTargetGroup]
    const targetGroup = TargetGroups[0];
    state.targetGroupArn = targetGroup.TargetGroupArn;
    state.targetGroupProtocol = targetGroup.Protocol;
    state.targetGroupPort = targetGroup.Port;
  }),
  new ScenarioOutput(
    "createdLoadBalancerTargetGroup",
    MESSAGES.createdLoadBalancerTargetGroup.replace(
      "${TARGET_GROUP_NAME}",
      NAMES.loadBalancerTargetGroupName,
    ),
  ),
  new ScenarioOutput(
    "creatingLoadBalancer",
    MESSAGES.creatingLoadBalancer.replace("${LB_NAME}", NAMES.loadBalancerName),
  ),
  new ScenarioAction("createLoadBalancer", async (state) => {
    // snippet-start:[javascript.v3.wkflw.resilient.CreateLoadBalancer]
    const client = new ElasticLoadBalancingV2Client({});
    const { LoadBalancers } = await client.send(
      new CreateLoadBalancerCommand({
        Name: NAMES.loadBalancerName,
        Subnets: state.subnets,
      }),
    );
    state.loadBalancerDns = LoadBalancers[0].DNSName;
    state.loadBalancerArn = LoadBalancers[0].LoadBalancerArn;
```

```
    await waitUntilLoadBalancerAvailable(
      { client },
      { Names: [NAMES.loadBalancerName] },
    );
    // snippet-end:[javascript.v3.wkflw.resilient.CreateLoadBalancer]
  }),
  new ScenarioOutput("createdLoadBalancer", (state) =>
    MESSAGES.createdLoadBalancer
      .replace("${LB_NAME}", NAMES.loadBalancerName)
      .replace("${DNS_NAME}", state.loadBalancerDns),
  ),
  new ScenarioOutput(
    "creatingListener",
    MESSAGES.creatingLoadBalancerListener
      .replace("${LB_NAME}", NAMES.loadBalancerName)
      .replace("${TARGET_GROUP_NAME}", NAMES.loadBalancerTargetGroupName),
  ),
  new ScenarioAction("createListener", async (state) => {
    // snippet-start:[javascript.v3.wkflw.resilient.CreateListener]
    const client = new ElasticLoadBalancingV2Client({});
    const { Listeners } = await client.send(
      new CreateListenerCommand({
        LoadBalancerArn: state.loadBalancerArn,
        Protocol: state.targetGroupProtocol,
        Port: state.targetGroupPort,
        DefaultActions: [
          { Type: "forward", TargetGroupArn: state.targetGroupArn },
        ],
      }),
    );
    // snippet-end:[javascript.v3.wkflw.resilient.CreateListener]
    const listener = Listeners[0];
    state.loadBalancerListenerArn = listener.ListenerArn;
  }),
  new ScenarioOutput("createdListener", (state) =>
    MESSAGES.createdLoadBalancerListener.replace(
      "${LB_LISTENER_ARN}",
      state.loadBalancerListenerArn,
    ),
  ),
  new ScenarioOutput(
    "attachingLoadBalancerTargetGroup",
    MESSAGES.attachingLoadBalancerTargetGroup
      .replace("${TARGET_GROUP_NAME}", NAMES.loadBalancerTargetGroupName)
```

```
      .replace("${AUTO_SCALING_GROUP_NAME}", NAMES.autoScalingGroupName),
  ),
  new ScenarioAction("attachLoadBalancerTargetGroup", async (state) => {
    // snippet-start:[javascript.v3.wkflw.resilient.AttachTargetGroup]
    const client = new AutoScalingClient({});
    await client.send(
      new AttachLoadBalancerTargetGroupsCommand({
        AutoScalingGroupName: NAMES.autoScalingGroupName,
        TargetGroupARNs: [state.targetGroupArn],
      }),
    );
    // snippet-end:[javascript.v3.wkflw.resilient.AttachTargetGroup]
  }),
  new ScenarioOutput(
    "attachedLoadBalancerTargetGroup",
    MESSAGES.attachedLoadBalancerTargetGroup,
  ),
  new ScenarioOutput("verifyingInboundPort", MESSAGES.verifyingInboundPort),
  new ScenarioAction(
    "verifyInboundPort",
    /**
     *
     * @param {{ defaultSecurityGroup: import('@aws-sdk/client-ec2').SecurityGroup}}
 state
     */
    async (state) => {
      const client = new EC2Client({});
      const { SecurityGroups } = await client.send(
        new DescribeSecurityGroupsCommand({
          Filters: [{ Name: "group-name", Values: ["default"] }],
        }),
      );
      if (!SecurityGroups) {
        state.verifyInboundPortError = new Error(MESSAGES.noSecurityGroups);
      }
      state.defaultSecurityGroup = SecurityGroups[0];

      /**
       * @type {string}
       */
      const ipResponse = (await axios.get("http://checkip.amazonaws.com")).data;
      state.myIp = ipResponse.trim();
      const myIpRules = state.defaultSecurityGroup.IpPermissions.filter(
        ({ IpRanges }) =>
```

```
          IpRanges.some(
            ({ CidrIp }) =>
              CidrIp.startsWith(state.myIp) || CidrIp === "0.0.0.0/0",
          ),
      )
        .filter(({ IpProtocol }) => IpProtocol === "tcp")
        .filter(({ FromPort }) => FromPort === 80);

      state.myIpRules = myIpRules;
    },
  ),
  new ScenarioOutput(
    "verifiedInboundPort",
    /**
     * @param {{ myIpRules: any[] }} state
     */
    (state) => {
      if (state.myIpRules.length > 0) {
        return MESSAGES.foundIpRules.replace(
          "${IP_RULES}",
          JSON.stringify(state.myIpRules, null, 2),
        );
      } else {
        return MESSAGES.noIpRules;
      }
    },
  ),
  new ScenarioInput(
    "shouldAddInboundRule",
    /**
     * @param {{ myIpRules: any[] }} state
     */
    (state) => {
      if (state.myIpRules.length > 0) {
        return false;
      } else {
        return MESSAGES.noIpRules;
      }
    },
    { type: "confirm" },
  ),
  new ScenarioAction(
    "addInboundRule",
    /**
```

```
     * @param {{ defaultSecurityGroup: import('@aws-sdk/client-
ec2').SecurityGroup }} state
     */
    async (state) => {
      if (!state.shouldAddInboundRule) {
        return;
      }

      const client = new EC2Client({});
      await client.send(
        new AuthorizeSecurityGroupIngressCommand({
          GroupId: state.defaultSecurityGroup.GroupId,
          CidrIp: `${state.myIp}/32`,
          FromPort: 80,
          ToPort: 80,
          IpProtocol: "tcp",
        }),
      );
    },
  ),
  new ScenarioOutput("addedInboundRule", (state) => {
    if (state.shouldAddInboundRule) {
      return MESSAGES.addedInboundRule.replace("${IP_ADDRESS}", state.myIp);
    } else {
      return false;
    }
  }),
  new ScenarioOutput("verifyingEndpoint", (state) =>
    MESSAGES.verifyingEndpoint.replace("${DNS_NAME}", state.loadBalancerDns),
  ),
  new ScenarioAction("verifyEndpoint", async (state) => {
    try {
      const response = await retry({ intervalInMs: 2000, maxRetries: 30 }, () =>
        axios.get(`http://${state.loadBalancerDns}`),
      );
      state.endpointResponse = JSON.stringify(response.data, null, 2);
    } catch (e) {
      state.verifyEndpointError = e;
    }
  }),
  new ScenarioOutput("verifiedEndpoint", (state) => {
    if (state.verifyEndpointError) {
      console.error(state.verifyEndpointError);
    } else {
```

```
      return MESSAGES.verifiedEndpoint.replace(
        "${ENDPOINT_RESPONSE}",
        state.endpointResponse,
      );
    }
  }),
];
```

创建运行演示的步骤。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { readFileSync } from "node:fs";
import { join } from "node:path";

import axios from "axios";

import {
  DescribeTargetGroupsCommand,
  DescribeTargetHealthCommand,
  ElasticLoadBalancingV2Client,
} from "@aws-sdk/client-elastic-load-balancing-v2";
import {
  DescribeInstanceInformationCommand,
  PutParameterCommand,
  SSMClient,
  SendCommandCommand,
} from "@aws-sdk/client-ssm";
import {
  IAMClient,
  CreatePolicyCommand,
  CreateRoleCommand,
  AttachRolePolicyCommand,
  CreateInstanceProfileCommand,
  AddRoleToInstanceProfileCommand,
  waitUntilInstanceProfileExists,
} from "@aws-sdk/client-iam";
import {
  AutoScalingClient,
  DescribeAutoScalingGroupsCommand,
  TerminateInstanceInAutoScalingGroupCommand,
} from "@aws-sdk/client-auto-scaling";
```

```
import {
  DescribeIamInstanceProfileAssociationsCommand,
  EC2Client,
  RebootInstancesCommand,
  ReplaceIamInstanceProfileAssociationCommand,
} from "@aws-sdk/client-ec2";

import {
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/scenario.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES, RESOURCES_PATH } from "./constants.js";
import { findLoadBalancer } from "./shared.js";

const getRecommendation = new ScenarioAction(
  "getRecommendation",
  async (state) => {
    const loadBalancer = await findLoadBalancer(NAMES.loadBalancerName);
    if (loadBalancer) {
      state.loadBalancerDnsName = loadBalancer.DNSName;
      try {
        state.recommendation = (
          await axios.get(`http://${state.loadBalancerDnsName}`)
        ).data;
      } catch (e) {
        state.recommendation = e instanceof Error ? e.message : e;
      }
    } else {
      throw new Error(MESSAGES.demoFindLoadBalancerError);
    }
  },
);

const getRecommendationResult = new ScenarioOutput(
  "getRecommendationResult",
  (state) =>
    `Recommendation:\n${JSON.stringify(state.recommendation, null, 2)}`,
  { preformatted: true },
);

const getHealthCheck = new ScenarioAction("getHealthCheck", async (state) => {
```

```
  // snippet-start:[javascript.v3.wkflw.resilient.DescribeTargetGroups]
  const client = new ElasticLoadBalancingV2Client({});
  const { TargetGroups } = await client.send(
    new DescribeTargetGroupsCommand({
      Names: [NAMES.loadBalancerTargetGroupName],
    }),
  );
  // snippet-end:[javascript.v3.wkflw.resilient.DescribeTargetGroups]

  // snippet-start:[javascript.v3.wkflw.resilient.DescribeTargetHealth]
  const { TargetHealthDescriptions } = await client.send(
    new DescribeTargetHealthCommand({
      TargetGroupArn: TargetGroups[0].TargetGroupArn,
    }),
  );
  // snippet-end:[javascript.v3.wkflw.resilient.DescribeTargetHealth]
  state.targetHealthDescriptions = TargetHealthDescriptions;
});

const getHealthCheckResult = new ScenarioOutput(
  "getHealthCheckResult",
  /**
   * @param {{ targetHealthDescriptions: import('@aws-sdk/client-elastic-load-
balancing-v2').TargetHealthDescription[]}} state
   */
  (state) => {
    const status = state.targetHealthDescriptions
      .map((th) => `${th.Target.Id}: ${th.TargetHealth.State}`)
      .join("\n");
    return `Health check:\n${status}`;
  },
  { preformatted: true },
);

const loadBalancerLoop = new ScenarioAction(
  "loadBalancerLoop",
  getRecommendation.action,
  {
    whileConfig: {
      inputEquals: true,
      input: new ScenarioInput(
        "loadBalancerCheck",
        MESSAGES.demoLoadBalancerCheck,
        {
```

```
          type: "confirm",
        },
      ),
      output: getRecommendationResult,
    },
  },
);

const healthCheckLoop = new ScenarioAction(
  "healthCheckLoop",
  getHealthCheck.action,
  {
    whileConfig: {
      inputEquals: true,
      input: new ScenarioInput("healthCheck", MESSAGES.demoHealthCheck, {
        type: "confirm",
      }),
      output: getHealthCheckResult,
    },
  },
);

const statusSteps = [
  getRecommendation,
  getRecommendationResult,
  getHealthCheck,
  getHealthCheckResult,
];

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const demoSteps = [
  new ScenarioOutput("header", MESSAGES.demoHeader, { header: true }),
  new ScenarioOutput("sanityCheck", MESSAGES.demoSanityCheck),
  ...statusSteps,
  new ScenarioInput(
    "brokenDependencyConfirmation",
    MESSAGES.demoBrokenDependencyConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("brokenDependency", async (state) => {
    if (!state.brokenDependencyConfirmation) {
      process.exit();
```

```
    } else {
      const client = new SSMClient({});
      state.badTableName = `fake-table-${Date.now()}`;
      await client.send(
        new PutParameterCommand({
          Name: NAMES.ssmTableNameKey,
          Value: state.badTableName,
          Overwrite: true,
          Type: "String",
        }),
      );
    }
  }),
  new ScenarioOutput("testBrokenDependency", (state) =>
    MESSAGES.demoTestBrokenDependency.replace(
      "${TABLE_NAME}",
      state.badTableName,
    ),
  ),
  ...statusSteps,
  new ScenarioInput(
    "staticResponseConfirmation",
    MESSAGES.demoStaticResponseConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("staticResponse", async (state) => {
    if (!state.staticResponseConfirmation) {
      process.exit();
    } else {
      const client = new SSMClient({});
      await client.send(
        new PutParameterCommand({
          Name: NAMES.ssmFailureResponseKey,
          Value: "static",
          Overwrite: true,
          Type: "String",
        }),
      );
    }
  }),
  new ScenarioOutput("testStaticResponse", MESSAGES.demoTestStaticResponse),
  ...statusSteps,
  new ScenarioInput(
    "badCredentialsConfirmation",
```

```
    MESSAGES.demoBadCredentialsConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("badCredentialsExit", (state) => {
    if (!state.badCredentialsConfirmation) {
      process.exit();
    }
  }),
  new ScenarioAction("fixDynamoDBName", async () => {
    const client = new SSMClient({});
    await client.send(
      new PutParameterCommand({
        Name: NAMES.ssmTableNameKey,
        Value: NAMES.tableName,
        Overwrite: true,
        Type: "String",
      }),
    );
  }),
  new ScenarioAction(
    "badCredentials",
    /**
     * @param {{ targetInstance: import('@aws-sdk/client-auto-scaling').Instance }}
 state
     */
    async (state) => {
      await createSsmOnlyInstanceProfile();
      const autoScalingClient = new AutoScalingClient({});
      const { AutoScalingGroups } = await autoScalingClient.send(
        new DescribeAutoScalingGroupsCommand({
          AutoScalingGroupNames: [NAMES.autoScalingGroupName],
        }),
      );
      state.targetInstance = AutoScalingGroups[0].Instances[0];
      // snippet-start:
[javascript.v3.wkflw.resilient.DescribeIamInstanceProfileAssociations]
      const ec2Client = new EC2Client({});
      const { IamInstanceProfileAssociations } = await ec2Client.send(
        new DescribeIamInstanceProfileAssociationsCommand({
          Filters: [
            { Name: "instance-id", Values: [state.targetInstance.InstanceId] },
          ],
        }),
      );
```

```
      // snippet-end:
[javascript.v3.wkflw.resilient.DescribeIamInstanceProfileAssociations]
      state.instanceProfileAssociationId =
        IamInstanceProfileAssociations[0].AssociationId;
      // snippet-start:
[javascript.v3.wkflw.resilient.ReplaceIamInstanceProfileAssociation]
      await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
        ec2Client.send(
          new ReplaceIamInstanceProfileAssociationCommand({
            AssociationId: state.instanceProfileAssociationId,
            IamInstanceProfile: { Name: NAMES.ssmOnlyInstanceProfileName },
          }),
        ),
      );
      // snippet-end:
[javascript.v3.wkflw.resilient.ReplaceIamInstanceProfileAssociation]

      await ec2Client.send(
        new RebootInstancesCommand({
          InstanceIds: [state.targetInstance.InstanceId],
        }),
      );

      const ssmClient = new SSMClient({});
      await retry({ intervalInMs: 20000, maxRetries: 15 }, async () => {
        const { InstanceInformationList } = await ssmClient.send(
          new DescribeInstanceInformationCommand({}),
        );

        const instance = InstanceInformationList.find(
          (info) => info.InstanceId === state.targetInstance.InstanceId,
        );

        if (!instance) {
          throw new Error("Instance not found.");
        }
      });

      await ssmClient.send(
        new SendCommandCommand({
          InstanceIds: [state.targetInstance.InstanceId],
          DocumentName: "AWS-RunShellScript",
          Parameters: { commands: ["cd / && sudo python3 server.py 80"] },
        }),
```

```
        );
      },
    ),
    new ScenarioOutput(
      "testBadCredentials",
      /**
       * @param {{ targetInstance: import('@aws-sdk/client-ssm').InstanceInformation}}
  state
       */
      (state) =>
        MESSAGES.demoTestBadCredentials.replace(
          "${INSTANCE_ID}",
          state.targetInstance.InstanceId,
        ),
    ),
    loadBalancerLoop,
    new ScenarioInput(
      "deepHealthCheckConfirmation",
      MESSAGES.demoDeepHealthCheckConfirmation,
      { type: "confirm" },
    ),
    new ScenarioAction("deepHealthCheckExit", (state) => {
      if (!state.deepHealthCheckConfirmation) {
        process.exit();
      }
    }),
    new ScenarioAction("deepHealthCheck", async () => {
      const client = new SSMClient({});
      await client.send(
        new PutParameterCommand({
          Name: NAMES.ssmHealthCheckKey,
          Value: "deep",
          Overwrite: true,
          Type: "String",
        }),
      );
    }),
    new ScenarioOutput("testDeepHealthCheck", MESSAGES.demoTestDeepHealthCheck),
    healthCheckLoop,
    loadBalancerLoop,
    new ScenarioInput(
      "killInstanceConfirmation",
      /**
```

```
     * @param {{ targetInstance: import('@aws-sdk/client-
ssm').InstanceInformation }} state
     */
    (state) =>
      MESSAGES.demoKillInstanceConfirmation.replace(
        "${INSTANCE_ID}",
        state.targetInstance.InstanceId,
      ),
    { type: "confirm" },
  ),
  new ScenarioAction("killInstanceExit", (state) => {
    if (!state.killInstanceConfirmation) {
      process.exit();
    }
  }),
  new ScenarioAction(
    "killInstance",
    /**
     * @param {{ targetInstance: import('@aws-sdk/client-
ssm').InstanceInformation }} state
     */
    async (state) => {
      const client = new AutoScalingClient({});
      await client.send(
        new TerminateInstanceInAutoScalingGroupCommand({
          InstanceId: state.targetInstance.InstanceId,
          ShouldDecrementDesiredCapacity: false,
        }),
      );
    },
  ),
  new ScenarioOutput("testKillInstance", MESSAGES.demoTestKillInstance),
  healthCheckLoop,
  loadBalancerLoop,
  new ScenarioInput("failOpenConfirmation", MESSAGES.demoFailOpenConfirmation, {
    type: "confirm",
  }),
  new ScenarioAction("failOpenExit", (state) => {
    if (!state.failOpenConfirmation) {
      process.exit();
    }
  }),
  new ScenarioAction("failOpen", () => {
    const client = new SSMClient({});
```

```
    return client.send(
      new PutParameterCommand({
        Name: NAMES.ssmTableNameKey,
        Value: `fake-table-${Date.now()}`,
        Overwrite: true,
        Type: "String",
      }),
    );
  }),
  new ScenarioOutput("testFailOpen", MESSAGES.demoFailOpenTest),
  healthCheckLoop,
  loadBalancerLoop,
  new ScenarioInput(
    "resetTableConfirmation",
    MESSAGES.demoResetTableConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("resetTableExit", (state) => {
    if (!state.resetTableConfirmation) {
      process.exit();
    }
  }),
  new ScenarioAction("resetTable", async () => {
    const client = new SSMClient({});
    await client.send(
      new PutParameterCommand({
        Name: NAMES.ssmTableNameKey,
        Value: NAMES.tableName,
        Overwrite: true,
        Type: "String",
      }),
    );
  }),
  new ScenarioOutput("testResetTable", MESSAGES.demoTestResetTable),
  healthCheckLoop,
  loadBalancerLoop,
];

async function createSsmOnlyInstanceProfile() {
  const iamClient = new IAMClient({});
  const { Policy } = await iamClient.send(
    new CreatePolicyCommand({
      PolicyName: NAMES.ssmOnlyPolicyName,
      PolicyDocument: readFileSync(
```

```
        join(RESOURCES_PATH, "ssm_only_policy.json"),
      ),
    }),
  );
  await iamClient.send(
    new CreateRoleCommand({
      RoleName: NAMES.ssmOnlyRoleName,
      AssumeRolePolicyDocument: JSON.stringify({
        Version: "2012-10-17",
        Statement: [
          {
            Effect: "Allow",
            Principal: { Service: "ec2.amazonaws.com" },
            Action: "sts:AssumeRole",
          },
        ],
      }),
    }),
  );
  await iamClient.send(
    new AttachRolePolicyCommand({
      RoleName: NAMES.ssmOnlyRoleName,
      PolicyArn: Policy.Arn,
    }),
  );
  await iamClient.send(
    new AttachRolePolicyCommand({
      RoleName: NAMES.ssmOnlyRoleName,
      PolicyArn: "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore",
    }),
  );
  // snippet-start:[javascript.v3.wkflw.resilient.CreateInstanceProfile]
  const { InstanceProfile } = await iamClient.send(
    new CreateInstanceProfileCommand({
      InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
    }),
  );
  await waitUntilInstanceProfileExists(
    { client: iamClient },
    { InstanceProfileName: NAMES.ssmOnlyInstanceProfileName },
  );
  // snippet-end:[javascript.v3.wkflw.resilient.CreateInstanceProfile]
  await iamClient.send(
    new AddRoleToInstanceProfileCommand({
```

```
      InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
      RoleName: NAMES.ssmOnlyRoleName,
    }),
  );

  return InstanceProfile;
}
```

创建销毁所有资源的步骤。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { unlinkSync } from "node:fs";

import { DynamoDBClient, DeleteTableCommand } from "@aws-sdk/client-dynamodb";
import {
  EC2Client,
  DeleteKeyPairCommand,
  DeleteLaunchTemplateCommand,
} from "@aws-sdk/client-ec2";
import {
  IAMClient,
  DeleteInstanceProfileCommand,
  RemoveRoleFromInstanceProfileCommand,
  DeletePolicyCommand,
  DeleteRoleCommand,
  DetachRolePolicyCommand,
  paginateListPolicies,
} from "@aws-sdk/client-iam";
import {
  AutoScalingClient,
  DeleteAutoScalingGroupCommand,
  TerminateInstanceInAutoScalingGroupCommand,
  UpdateAutoScalingGroupCommand,
  paginateDescribeAutoScalingGroups,
} from "@aws-sdk/client-auto-scaling";
import {
  DeleteLoadBalancerCommand,
  DeleteTargetGroupCommand,
  DescribeTargetGroupsCommand,
  ElasticLoadBalancingV2Client,
} from "@aws-sdk/client-elastic-load-balancing-v2";
```

```
import {
  ScenarioOutput,
  ScenarioInput,
  ScenarioAction,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES } from "./constants.js";
import { findLoadBalancer } from "./shared.js";

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const destroySteps = [
  new ScenarioInput("destroy", MESSAGES.destroy, { type: "confirm" }),
  new ScenarioAction(
    "abort",
    (state) => state.destroy === false && process.exit(),
  ),
  new ScenarioAction("deleteTable", async (c) => {
    try {
      const client = new DynamoDBClient({});
      await client.send(new DeleteTableCommand({ TableName: NAMES.tableName }));
    } catch (e) {
      c.deleteTableError = e;
    }
  }),
  new ScenarioOutput("deleteTableResult", (state) => {
    if (state.deleteTableError) {
      console.error(state.deleteTableError);
      return MESSAGES.deleteTableError.replace(
        "${TABLE_NAME}",
        NAMES.tableName,
      );
    } else {
      return MESSAGES.deletedTable.replace("${TABLE_NAME}", NAMES.tableName);
    }
  }),
  new ScenarioAction("deleteKeyPair", async (state) => {
    try {
      const client = new EC2Client({});
      await client.send(
        new DeleteKeyPairCommand({ KeyName: NAMES.keyPairName }),
```

```
      );
      unlinkSync(`${NAMES.keyPairName}.pem`);
    } catch (e) {
      state.deleteKeyPairError = e;
    }
  }),
  new ScenarioOutput("deleteKeyPairResult", (state) => {
    if (state.deleteKeyPairError) {
      console.error(state.deleteKeyPairError);
      return MESSAGES.deleteKeyPairError.replace(
        "${KEY_PAIR_NAME}",
        NAMES.keyPairName,
      );
    } else {
      return MESSAGES.deletedKeyPair.replace(
        "${KEY_PAIR_NAME}",
        NAMES.keyPairName,
      );
    }
  }),
  new ScenarioAction("detachPolicyFromRole", async (state) => {
    try {
      const client = new IAMClient({});
      const policy = await findPolicy(NAMES.instancePolicyName);

      if (!policy) {
        state.detachPolicyFromRoleError = new Error(
          `Policy ${NAMES.instancePolicyName} not found.`,
        );
      } else {
        await client.send(
          new DetachRolePolicyCommand({
            RoleName: NAMES.instanceRoleName,
            PolicyArn: policy.Arn,
          }),
        );
      }
    } catch (e) {
      state.detachPolicyFromRoleError = e;
    }
  }),
  new ScenarioOutput("detachedPolicyFromRole", (state) => {
    if (state.detachPolicyFromRoleError) {
      console.error(state.detachPolicyFromRoleError);
```

```
      return MESSAGES.detachPolicyFromRoleError
        .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    } else {
      return MESSAGES.detachedPolicyFromRole
        .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    }
  }),
  new ScenarioAction("deleteInstancePolicy", async (state) => {
    const client = new IAMClient({});
    const policy = await findPolicy(NAMES.instancePolicyName);

    if (!policy) {
      state.deletePolicyError = new Error(
        `Policy ${NAMES.instancePolicyName} not found.`,
      );
    } else {
      return client.send(
        new DeletePolicyCommand({
          PolicyArn: policy.Arn,
        }),
      );
    }
  }),
  new ScenarioOutput("deletePolicyResult", (state) => {
    if (state.deletePolicyError) {
      console.error(state.deletePolicyError);
      return MESSAGES.deletePolicyError.replace(
        "${INSTANCE_POLICY_NAME}",
        NAMES.instancePolicyName,
      );
    } else {
      return MESSAGES.deletedPolicy.replace(
        "${INSTANCE_POLICY_NAME}",
        NAMES.instancePolicyName,
      );
    }
  }),
  new ScenarioAction("removeRoleFromInstanceProfile", async (state) => {
    try {
      const client = new IAMClient({});
      await client.send(
        new RemoveRoleFromInstanceProfileCommand({
```

```
          RoleName: NAMES.instanceRoleName,
          InstanceProfileName: NAMES.instanceProfileName,
        }),
      );
    } catch (e) {
      state.removeRoleFromInstanceProfileError = e;
    }
  }),
  new ScenarioOutput("removeRoleFromInstanceProfileResult", (state) => {
    if (state.removeRoleFromInstanceProfile) {
      console.error(state.removeRoleFromInstanceProfileError);
      return MESSAGES.removeRoleFromInstanceProfileError
        .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    } else {
      return MESSAGES.removedRoleFromInstanceProfile
        .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    }
  }),
  new ScenarioAction("deleteInstanceRole", async (state) => {
    try {
      const client = new IAMClient({});
      await client.send(
        new DeleteRoleCommand({
          RoleName: NAMES.instanceRoleName,
        }),
      );
    } catch (e) {
      state.deleteInstanceRoleError = e;
    }
  }),
  new ScenarioOutput("deleteInstanceRoleResult", (state) => {
    if (state.deleteInstanceRoleError) {
      console.error(state.deleteInstanceRoleError);
      return MESSAGES.deleteInstanceRoleError.replace(
        "${INSTANCE_ROLE_NAME}",
        NAMES.instanceRoleName,
      );
    } else {
      return MESSAGES.deletedInstanceRole.replace(
        "${INSTANCE_ROLE_NAME}",
        NAMES.instanceRoleName,
      );
```

```
    }
  }),
  new ScenarioAction("deleteInstanceProfile", async (state) => {
    try {
      // snippet-start:[javascript.v3.wkflw.resilient.DeleteInstanceProfile]
      const client = new IAMClient({});
      await client.send(
        new DeleteInstanceProfileCommand({
          InstanceProfileName: NAMES.instanceProfileName,
        }),
      );
      // snippet-end:[javascript.v3.wkflw.resilient.DeleteInstanceProfile]
    } catch (e) {
      state.deleteInstanceProfileError = e;
    }
  }),
  new ScenarioOutput("deleteInstanceProfileResult", (state) => {
    if (state.deleteInstanceProfileError) {
      console.error(state.deleteInstanceProfileError);
      return MESSAGES.deleteInstanceProfileError.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.instanceProfileName,
      );
    } else {
      return MESSAGES.deletedInstanceProfile.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.instanceProfileName,
      );
    }
  }),
  new ScenarioAction("deleteAutoScalingGroup", async (state) => {
    try {
      await terminateGroupInstances(NAMES.autoScalingGroupName);
      await retry({ intervalInMs: 60000, maxRetries: 60 }, async () => {
        await deleteAutoScalingGroup(NAMES.autoScalingGroupName);
      });
    } catch (e) {
      state.deleteAutoScalingGroupError = e;
    }
  }),
  new ScenarioOutput("deleteAutoScalingGroupResult", (state) => {
    if (state.deleteAutoScalingGroupError) {
      console.error(state.deleteAutoScalingGroupError);
      return MESSAGES.deleteAutoScalingGroupError.replace(
```

```
        "${AUTO_SCALING_GROUP_NAME}",
        NAMES.autoScalingGroupName,
      );
    } else {
      return MESSAGES.deletedAutoScalingGroup.replace(
        "${AUTO_SCALING_GROUP_NAME}",
        NAMES.autoScalingGroupName,
      );
    }
  }),
  new ScenarioAction("deleteLaunchTemplate", async (state) => {
    const client = new EC2Client({});
    try {
      // snippet-start:[javascript.v3.wkflw.resilient.DeleteLaunchTemplate]
      await client.send(
        new DeleteLaunchTemplateCommand({
          LaunchTemplateName: NAMES.launchTemplateName,
        }),
      );
      // snippet-end:[javascript.v3.wkflw.resilient.DeleteLaunchTemplate]
    } catch (e) {
      state.deleteLaunchTemplateError = e;
    }
  }),
  new ScenarioOutput("deleteLaunchTemplateResult", (state) => {
    if (state.deleteLaunchTemplateError) {
      console.error(state.deleteLaunchTemplateError);
      return MESSAGES.deleteLaunchTemplateError.replace(
        "${LAUNCH_TEMPLATE_NAME}",
        NAMES.launchTemplateName,
      );
    } else {
      return MESSAGES.deletedLaunchTemplate.replace(
        "${LAUNCH_TEMPLATE_NAME}",
        NAMES.launchTemplateName,
      );
    }
  }),
  new ScenarioAction("deleteLoadBalancer", async (state) => {
    try {
      // snippet-start:[javascript.v3.wkflw.resilient.DeleteLoadBalancer]
      const client = new ElasticLoadBalancingV2Client({});
      const loadBalancer = await findLoadBalancer(NAMES.loadBalancerName);
      await client.send(
```

```
        new DeleteLoadBalancerCommand({
          LoadBalancerArn: loadBalancer.LoadBalancerArn,
        }),
      );
      await retry({ intervalInMs: 1000, maxRetries: 60 }, async () => {
        const lb = await findLoadBalancer(NAMES.loadBalancerName);
        if (lb) {
          throw new Error("Load balancer still exists.");
        }
      });
      // snippet-end:[javascript.v3.wkflw.resilient.DeleteLoadBalancer]
    } catch (e) {
      state.deleteLoadBalancerError = e;
    }
  }),
  new ScenarioOutput("deleteLoadBalancerResult", (state) => {
    if (state.deleteLoadBalancerError) {
      console.error(state.deleteLoadBalancerError);
      return MESSAGES.deleteLoadBalancerError.replace(
        "${LB_NAME}",
        NAMES.loadBalancerName,
      );
    } else {
      return MESSAGES.deletedLoadBalancer.replace(
        "${LB_NAME}",
        NAMES.loadBalancerName,
      );
    }
  }),
  new ScenarioAction("deleteLoadBalancerTargetGroup", async (state) => {
    // snippet-start:[javascript.v3.wkflw.resilient.DeleteTargetGroup]
    const client = new ElasticLoadBalancingV2Client({});
    try {
      const { TargetGroups } = await client.send(
        new DescribeTargetGroupsCommand({
          Names: [NAMES.loadBalancerTargetGroupName],
        }),
      );

      await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
        client.send(
          new DeleteTargetGroupCommand({
            TargetGroupArn: TargetGroups[0].TargetGroupArn,
          }),
```

```
        ),
      );
    } catch (e) {
      state.deleteLoadBalancerTargetGroupError = e;
    }
    // snippet-end:[javascript.v3.wkflw.resilient.DeleteTargetGroup]
  }),
  new ScenarioOutput("deleteLoadBalancerTargetGroupResult", (state) => {
    if (state.deleteLoadBalancerTargetGroupError) {
      console.error(state.deleteLoadBalancerTargetGroupError);
      return MESSAGES.deleteLoadBalancerTargetGroupError.replace(
        "${TARGET_GROUP_NAME}",
        NAMES.loadBalancerTargetGroupName,
      );
    } else {
      return MESSAGES.deletedLoadBalancerTargetGroup.replace(
        "${TARGET_GROUP_NAME}",
        NAMES.loadBalancerTargetGroupName,
      );
    }
  }),
  new ScenarioAction("detachSsmOnlyRoleFromProfile", async (state) => {
    try {
      const client = new IAMClient({});
      await client.send(
        new RemoveRoleFromInstanceProfileCommand({
          InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
          RoleName: NAMES.ssmOnlyRoleName,
        }),
      );
    } catch (e) {
      state.detachSsmOnlyRoleFromProfileError = e;
    }
  }),
  new ScenarioOutput("detachSsmOnlyRoleFromProfileResult", (state) => {
    if (state.detachSsmOnlyRoleFromProfileError) {
      console.error(state.detachSsmOnlyRoleFromProfileError);
      return MESSAGES.detachSsmOnlyRoleFromProfileError
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${PROFILE_NAME}", NAMES.ssmOnlyInstanceProfileName);
    } else {
      return MESSAGES.detachedSsmOnlyRoleFromProfile
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${PROFILE_NAME}", NAMES.ssmOnlyInstanceProfileName);
```

```
    }
  }),
  new ScenarioAction("detachSsmOnlyCustomRolePolicy", async (state) => {
    try {
      const iamClient = new IAMClient({});
      const ssmOnlyPolicy = await findPolicy(NAMES.ssmOnlyPolicyName);
      await iamClient.send(
        new DetachRolePolicyCommand({
          RoleName: NAMES.ssmOnlyRoleName,
          PolicyArn: ssmOnlyPolicy.Arn,
        }),
      );
    } catch (e) {
      state.detachSsmOnlyCustomRolePolicyError = e;
    }
  }),
  new ScenarioOutput("detachSsmOnlyCustomRolePolicyResult", (state) => {
    if (state.detachSsmOnlyCustomRolePolicyError) {
      console.error(state.detachSsmOnlyCustomRolePolicyError);
      return MESSAGES.detachSsmOnlyCustomRolePolicyError
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${POLICY_NAME}", NAMES.ssmOnlyPolicyName);
    } else {
      return MESSAGES.detachedSsmOnlyCustomRolePolicy
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${POLICY_NAME}", NAMES.ssmOnlyPolicyName);
    }
  }),
  new ScenarioAction("detachSsmOnlyAWSRolePolicy", async (state) => {
    try {
      const iamClient = new IAMClient({});
      await iamClient.send(
        new DetachRolePolicyCommand({
          RoleName: NAMES.ssmOnlyRoleName,
          PolicyArn: "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore",
        }),
      );
    } catch (e) {
      state.detachSsmOnlyAWSRolePolicyError = e;
    }
  }),
  new ScenarioOutput("detachSsmOnlyAWSRolePolicyResult", (state) => {
    if (state.detachSsmOnlyAWSRolePolicyError) {
      console.error(state.detachSsmOnlyAWSRolePolicyError);
```

```
      return MESSAGES.detachSsmOnlyAWSRolePolicyError
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${POLICY_NAME}", "AmazonSSMManagedInstanceCore");
    } else {
      return MESSAGES.detachedSsmOnlyAWSRolePolicy
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${POLICY_NAME}", "AmazonSSMManagedInstanceCore");
    }
  }),
  new ScenarioAction("deleteSsmOnlyInstanceProfile", async (state) => {
    try {
      const iamClient = new IAMClient({});
      await iamClient.send(
        new DeleteInstanceProfileCommand({
          InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
        }),
      );
    } catch (e) {
      state.deleteSsmOnlyInstanceProfileError = e;
    }
  }),
  new ScenarioOutput("deleteSsmOnlyInstanceProfileResult", (state) => {
    if (state.deleteSsmOnlyInstanceProfileError) {
      console.error(state.deleteSsmOnlyInstanceProfileError);
      return MESSAGES.deleteSsmOnlyInstanceProfileError.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.ssmOnlyInstanceProfileName,
      );
    } else {
      return MESSAGES.deletedSsmOnlyInstanceProfile.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.ssmOnlyInstanceProfileName,
      );
    }
  }),
  new ScenarioAction("deleteSsmOnlyPolicy", async (state) => {
    try {
      const iamClient = new IAMClient({});
      const ssmOnlyPolicy = await findPolicy(NAMES.ssmOnlyPolicyName);
      await iamClient.send(
        new DeletePolicyCommand({
          PolicyArn: ssmOnlyPolicy.Arn,
        }),
      );
```

```
    } catch (e) {
      state.deleteSsmOnlyPolicyError = e;
    }
  }),
  new ScenarioOutput("deleteSsmOnlyPolicyResult", (state) => {
    if (state.deleteSsmOnlyPolicyError) {
      console.error(state.deleteSsmOnlyPolicyError);
      return MESSAGES.deleteSsmOnlyPolicyError.replace(
        "${POLICY_NAME}",
        NAMES.ssmOnlyPolicyName,
      );
    } else {
      return MESSAGES.deletedSsmOnlyPolicy.replace(
        "${POLICY_NAME}",
        NAMES.ssmOnlyPolicyName,
      );
    }
  }),
  new ScenarioAction("deleteSsmOnlyRole", async (state) => {
    try {
      const iamClient = new IAMClient({});
      await iamClient.send(
        new DeleteRoleCommand({
          RoleName: NAMES.ssmOnlyRoleName,
        }),
      );
    } catch (e) {
      state.deleteSsmOnlyRoleError = e;
    }
  }),
  new ScenarioOutput("deleteSsmOnlyRoleResult", (state) => {
    if (state.deleteSsmOnlyRoleError) {
      console.error(state.deleteSsmOnlyRoleError);
      return MESSAGES.deleteSsmOnlyRoleError.replace(
        "${ROLE_NAME}",
        NAMES.ssmOnlyRoleName,
      );
    } else {
      return MESSAGES.deletedSsmOnlyRole.replace(
        "${ROLE_NAME}",
        NAMES.ssmOnlyRoleName,
      );
    }
  }),
```

```
];

/**
 * @param {string} policyName
 */
async function findPolicy(policyName) {
  const client = new IAMClient({});
  const paginatedPolicies = paginateListPolicies({ client }, {});
  for await (const page of paginatedPolicies) {
    const policy = page.Policies.find((p) => p.PolicyName === policyName);
    if (policy) {
      return policy;
    }
  }
}

/**
 * @param {string} groupName
 */
async function deleteAutoScalingGroup(groupName) {
  const client = new AutoScalingClient({});
  try {
    await client.send(
      new DeleteAutoScalingGroupCommand({
        AutoScalingGroupName: groupName,
      }),
    );
  } catch (err) {
    if (!(err instanceof Error)) {
      throw err;
    } else {
      console.log(err.name);
      throw err;
    }
  }
}

/**
 * @param {string} groupName
 */
async function terminateGroupInstances(groupName) {
  const autoScalingClient = new AutoScalingClient({});
  const group = await findAutoScalingGroup(groupName);
  await autoScalingClient.send(
```

```
      new UpdateAutoScalingGroupCommand({
        AutoScalingGroupName: group.AutoScalingGroupName,
        MinSize: 0,
      }),
    );
    for (const i of group.Instances) {
      await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
        autoScalingClient.send(
          new TerminateInstanceInAutoScalingGroupCommand({
            InstanceId: i.InstanceId,
            ShouldDecrementDesiredCapacity: true,
          }),
        ),
      );
    }
}

async function findAutoScalingGroup(groupName) {
  const client = new AutoScalingClient({});
  const paginatedGroups = paginateDescribeAutoScalingGroups({ client }, {});
  for await (const page of paginatedGroups) {
    const group = page.AutoScalingGroups.find(
      (g) => g.AutoScalingGroupName === groupName,
    );
    if (group) {
      return group;
    }
  }
  throw new Error(`Auto scaling group ${groupName} not found.`);
}
```

- 有关 API 详细信息，请参阅《Amazon SDK for JavaScript API 参考》中的以下主题。
  - AttachLoadBalancerTargetGroups
  - CreateAutoScalingGroup
  - CreateInstanceProfile
  - CreateLaunchTemplate
  - CreateListener
  - CreateLoadBalancer
  - CreateTargetGroup

- DeleteAutoScalingGroup
- DeleteInstanceProfile
- DeleteLaunchTemplate
- DeleteLoadBalancer
- DeleteTargetGroup
- DescribeAutoScalingGroups
- DescribeAvailabilityZones
- DescribeIamInstanceProfileAssociations
- DescribeInstances
- DescribeLoadBalancers
- DescribeSubnets
- DescribeTargetGroups
- DescribeTargetHealth
- DescribeVpcs
- RebootInstances
- ReplaceIamInstanceProfileAssociation
- TerminateInstanceInAutoScalingGroup
- UpdateAutoScalingGroup

开始使用实例

以下代码示例演示了操作流程：

- 创建密钥对和安全组。
- 选择 Amazon 机器映像 (AMI) 和兼容的实例类型，然后创建实例。
- 停止实例，然后再重启。
- 将弹性 IP 地址与您的实例相关联。
- 使用 SSH 连接到您的实例，然后清理资源。

适用于 JavaScript (v3) 的软件开发工具包

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

在命令提示符中运行交互式场景。

```
import { mkdtempSync, writeFileSync, rmSync } from "fs";
import { tmpdir } from "os";
import { join } from "path";
import { get } from "http";

import {
  AllocateAddressCommand,
  AssociateAddressCommand,
  AuthorizeSecurityGroupIngressCommand,
  CreateKeyPairCommand,
  CreateSecurityGroupCommand,
  DeleteKeyPairCommand,
  DeleteSecurityGroupCommand,
  DescribeInstancesCommand,
  DescribeKeyPairsCommand,
  DescribeSecurityGroupsCommand,
  DisassociateAddressCommand,
  EC2Client,
  paginateDescribeImages,
  paginateDescribeInstanceTypes,
  ReleaseAddressCommand,
  RunInstancesCommand,
  StartInstancesCommand,
  StopInstancesCommand,
  TerminateInstancesCommand,
  waitUntilInstanceStatusOk,
  waitUntilInstanceStopped,
  waitUntilInstanceTerminated,
} from "@aws-sdk/client-ec2";
import { paginateGetParametersByPath, SSMClient } from "@aws-sdk/client-ssm";

import { wrapText } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { Prompter } from "@aws-doc-sdk-examples/lib/prompter.js";
```

```
const ec2Client = new EC2Client();
const ssmClient = new SSMClient();

const prompter = new Prompter();
const confirmMessage = "Continue?";
const tmpDirectory = mkdtempSync(join(tmpdir(), "ec2-scenario-tmp"));

const createKeyPair = async (keyPairName) => {
  // Create a key pair in Amazon EC2.
  const { KeyMaterial, KeyPairId } = await ec2Client.send(
    // A unique name for the key pair. Up to 255 ASCII characters.
    new CreateKeyPairCommand({ KeyName: keyPairName }),
  );

  // Save the private key in a temporary location.
  writeFileSync(`${tmpDirectory}/${keyPairName}.pem`, KeyMaterial, {
    mode: 0o400,
  });

  return KeyPairId;
};

const describeKeyPair = async (keyPairName) => {
  const command = new DescribeKeyPairsCommand({
    KeyNames: [keyPairName],
  });
  const { KeyPairs } = await ec2Client.send(command);
  return KeyPairs[0];
};

const createSecurityGroup = async (securityGroupName) => {
  const command = new CreateSecurityGroupCommand({
    GroupName: securityGroupName,
    Description: "A security group for the Amazon EC2 example.",
  });
  const { GroupId } = await ec2Client.send(command);
  return GroupId;
};

const allocateIpAddress = async () => {
  const command = new AllocateAddressCommand({});
  const { PublicIp, AllocationId } = await ec2Client.send(command);
  return { PublicIp, AllocationId };
```

```
};

const getLocalIpAddress = () => {
  return new Promise((res, rej) => {
    get("http://checkip.amazonaws.com", (response) => {
      let data = "";
      response.on("data", (chunk) => (data += chunk));
      response.on("end", () => res(data.trim()));
    }).on("error", (err) => {
      rej(err);
    });
  });
};

const authorizeSecurityGroupIngress = async (securityGroupId) => {
  const ipAddress = await getLocalIpAddress();
  const command = new AuthorizeSecurityGroupIngressCommand({
    GroupId: securityGroupId,
    IpPermissions: [
      {
        IpProtocol: "tcp",
        FromPort: 22,
        ToPort: 22,
        IpRanges: [{ CidrIp: `${ipAddress}/32` }],
      },
    ],
  });

  await ec2Client.send(command);
  return ipAddress;
};

const describeSecurityGroup = async (securityGroupName) => {
  const command = new DescribeSecurityGroupsCommand({
    GroupNames: [securityGroupName],
  });
  const { SecurityGroups } = await ec2Client.send(command);

  return SecurityGroups[0];
};

const getAmznLinux2AMIs = async () => {
  const AMIs = [];
  for await (const page of paginateGetParametersByPath(
```

```
    {
      client: ssmClient,
    },
    { Path: "/aws/service/ami-amazon-linux-latest" },
  )) {
    page.Parameters.forEach((param) => {
      if (param.Name.includes("amzn2")) {
        AMIs.push(param.Value);
      }
    });
  }

  const imageDetails = [];

  for await (const page of paginateDescribeImages(
    { client: ec2Client },
    { ImageIds: AMIs },
  )) {
    imageDetails.push(...(page.Images || []));
  }

  const choices = imageDetails.map((image, index) => ({
    name: `${image.ImageId} - ${image.Description}`,
    value: index,
  }));

  /**
   * @type {number}
   */
  const selectedIndex = await prompter.select({
    message: "Select an image.",
    choices,
  });

  return imageDetails[selectedIndex];
};

/**
 * @param {import('@aws-sdk/client-ec2').Image} imageDetails
 */
const getCompatibleInstanceTypes = async (imageDetails) => {
  const paginator = paginateDescribeInstanceTypes(
    { client: ec2Client, pageSize: 25 },
    {
```

```
      Filters: [
        {
          Name: "processor-info.supported-architecture",
          Values: [imageDetails.Architecture],
        },
        { Name: "instance-type", Values: ["*.micro", "*.small"] },
      ],
    },
  );

  const instanceTypes = [];

  for await (const page of paginator) {
    if (page.InstanceTypes.length) {
      instanceTypes.push(...(page.InstanceTypes || []));
    }
  }

  const choices = instanceTypes.map((type, index) => ({
    name: `${type.InstanceType} - Memory:${type.MemoryInfo.SizeInMiB}`,
    value: index,
  }));

  /**
   * @type {number}
   */
  const selectedIndex = await prompter.select({
    message: "Select an instance type.",
    choices,
  });
  return instanceTypes[selectedIndex];
};

const runInstance = async ({
  keyPairName,
  securityGroupId,
  imageId,
  instanceType,
}) => {
  const command = new RunInstancesCommand({
    KeyName: keyPairName,
    SecurityGroupIds: [securityGroupId],
    ImageId: imageId,
    InstanceType: instanceType,
```

```
    MinCount: 1,
    MaxCount: 1,
  });

  const { Instances } = await ec2Client.send(command);
  await waitUntilInstanceStatusOk(
    { client: ec2Client },
    { InstanceIds: [Instances[0].InstanceId] },
  );
  return Instances[0].InstanceId;
};

const describeInstance = async (instanceId) => {
  const command = new DescribeInstancesCommand({
    InstanceIds: [instanceId],
  });

  const { Reservations } = await ec2Client.send(command);
  return Reservations[0].Instances[0];
};

const displaySSHConnectionInfo = ({ publicIp, keyPairName }) => {
  return `ssh -i ${tmpDirectory}/${keyPairName}.pem ec2-user@${publicIp}`;
};

const stopInstance = async (instanceId) => {
  const command = new StopInstancesCommand({ InstanceIds: [instanceId] });
  await ec2Client.send(command);
  await waitUntilInstanceStopped(
    { client: ec2Client },
    { InstanceIds: [instanceId] },
  );
};

const startInstance = async (instanceId) => {
  const startCommand = new StartInstancesCommand({ InstanceIds: [instanceId] });
  await ec2Client.send(startCommand);
  await waitUntilInstanceStatusOk(
    { client: ec2Client },
    { InstanceIds: [instanceId] },
  );
  return await describeInstance(instanceId);
};
```

```
const associateAddress = async ({ allocationId, instanceId }) => {
  const command = new AssociateAddressCommand({
    AllocationId: allocationId,
    InstanceId: instanceId,
  });

  const { AssociationId } = await ec2Client.send(command);
  return AssociationId;
};

const disassociateAddress = async (associationId) => {
  const command = new DisassociateAddressCommand({
    AssociationId: associationId,
  });
  try {
    await ec2Client.send(command);
  } catch (err) {
    console.warn(
      `Failed to disassociated address with association id: ${associationId}`,
      err,
    );
  }
};

const releaseAddress = async (allocationId) => {
  const command = new ReleaseAddressCommand({
    AllocationId: allocationId,
  });

  try {
    await ec2Client.send(command);
    console.log(`Address with allocation ID ${allocationId} released.\n`);
  } catch (err) {
    console.log(
      `Failed to release address with allocation id: ${allocationId}.`,
      err,
    );
  }
};

const restartInstance = async (instanceId) => {
  console.log("Stopping instance.");
  await stopInstance(instanceId);
  console.log("Instance stopped.");
```

```
  console.log("Starting instance.");
  const { PublicIpAddress } = await startInstance(instanceId);
  return PublicIpAddress;
};

const terminateInstance = async (instanceId) => {
  const command = new TerminateInstancesCommand({
    InstanceIds: [instanceId],
  });

  try {
    await ec2Client.send(command);
    await waitUntilInstanceTerminated(
      { client: ec2Client },
      { InstanceIds: [instanceId] },
    );
    console.log(`Instance with ID ${instanceId} terminated.\n`);
  } catch (err) {
    console.warn(`Failed to terminate instance ${instanceId}.`, err);
  }
};

const deleteSecurityGroup = async (securityGroupId) => {
  const command = new DeleteSecurityGroupCommand({
    GroupId: securityGroupId,
  });

  try {
    await ec2Client.send(command);
    console.log(`Security group ${securityGroupId} deleted.\n`);
  } catch (err) {
    console.warn(`Failed to delete security group ${securityGroupId}.`, err);
  }
};

const deleteKeyPair = async (keyPairName) => {
  const command = new DeleteKeyPairCommand({
    KeyName: keyPairName,
  });

  try {
    await ec2Client.send(command);
    console.log(`Key pair ${keyPairName} deleted.\n`);
  } catch (err) {
```

```
    console.warn(`Failed to delete key pair ${keyPairName}.`, err);
  }
};

const deleteTemporaryDirectory = () => {
  try {
    rmSync(tmpDirectory, { recursive: true });
    console.log(`Temporary directory ${tmpDirectory} deleted.\n`);
  } catch (err) {
    console.warn(`Failed to delete temporary directory ${tmpDirectory}.`, err);
  }
};

export const main = async () => {
  const keyPairName = "ec2-scenario-key-pair";
  const securityGroupName = "ec2-scenario-security-group";

  let securityGroupId, ipAllocationId, publicIp, instanceId, associationId;

  console.log(wrapText("Welcome to the Amazon EC2 basic usage scenario."));

  try {
    // Prerequisites
    console.log(
      "Before you launch an instance, you'll need a few things:",
      "\n - A Key Pair",
      "\n - A Security Group",
      "\n - An IP Address",
      "\n - An AMI",
      "\n - A compatible instance type",
      "\n\n I'll go ahead and take care of the first three, but I'll need your help
 for the rest.",
    );

    await prompter.confirm({ message: confirmMessage });

    await createKeyPair(keyPairName);
    securityGroupId = await createSecurityGroup(securityGroupName);
    const { PublicIp, AllocationId } = await allocateIpAddress();
    ipAllocationId = AllocationId;
    publicIp = PublicIp;
    const ipAddress = await authorizeSecurityGroupIngress(securityGroupId);

    const { KeyName } = await describeKeyPair(keyPairName);
```

```
    const { GroupName } = await describeSecurityGroup(securityGroupName);
    console.log(`# created the key pair ${KeyName}.\n`);
    console.log(
      `# created the security group ${GroupName}`,
      `and allowed SSH access from ${ipAddress} (your IP).\n`,
    );
    console.log(`# allocated ${publicIp} to be used for your EC2 instance.\n`);

    await prompter.confirm({ message: confirmMessage });

    // Creating the instance
    console.log(wrapText("Create the instance."));
    console.log(
      "You get to choose which image you want. Select an amazon-linux-2 image from
 the following:",
    );
    const imageDetails = await getAmznLinux2AMIs();
    const instanceTypeDetails = await getCompatibleInstanceTypes(imageDetails);
    console.log("Creating your instance. This can take a few seconds.");
    instanceId = await runInstance({
      keyPairName,
      securityGroupId,
      imageId: imageDetails.ImageId,
      instanceType: instanceTypeDetails.InstanceType,
    });
    const instanceDetails = await describeInstance(instanceId);
    console.log(`# instance ${instanceId}.\n`);
    console.log(instanceDetails);
    console.log(
      `\nYou should now be able to SSH into your instance from another terminal:`,
      `\n${displaySSHConnectionInfo({
        publicIp: instanceDetails.PublicIpAddress,
        keyPairName,
      })}`,
    );

    await prompter.confirm({ message: confirmMessage });

    // Understanding the IP address.
    console.log(wrapText("Understanding the IP address."));
    console.log(
      "When you stop and start an instance, the IP address will change. I'll restart
 your",
      "instance for you. Notice how the IP address changes.",
```

```
    );
    const ipAddressAfterRestart = await restartInstance(instanceId);
    console.log(
      `\n Instance started. The IP address changed from
${instanceDetails.PublicIpAddress} to ${ipAddressAfterRestart}`,
      `\n${displaySSHConnectionInfo({
        publicIp: ipAddressAfterRestart,
        keyPairName,
      })}`,
    );
    await prompter.confirm({ message: confirmMessage });
    console.log(
      `If you want to the IP address to be static, you can associate an allocated`,
      `IP address to your instance. I allocated ${publicIp} for you earlier, and now
I'll associate it to your instance.`,
    );
    associationId = await associateAddress({
      allocationId: ipAllocationId,
      instanceId,
    });
    console.log(
      "Done. Now you should be able to SSH using the new IP.\n",
      `${displaySSHConnectionInfo({ publicIp, keyPairName })}`,
    );
    await prompter.confirm({ message: confirmMessage });
    console.log(
      "I'll restart the server again so you can see the IP address remains the
same.",
    );
    const ipAddressAfterAssociated = await restartInstance(instanceId);
    console.log(
      `Done. Here's your SSH info. Notice the IP address hasn't changed.`,
      `\n${displaySSHConnectionInfo({
        publicIp: ipAddressAfterAssociated,
        keyPairName,
      })}`,
    );
    await prompter.confirm({ message: confirmMessage });
  } catch (err) {
    console.error(err);
  } finally {
    // Clean up.
    console.log(wrapText("Clean up."));
    console.log("Now I'll clean up all of the stuff I created.");
```

```
    await prompter.confirm({ message: confirmMessage });
    console.log("Cleaning up. Some of these steps can take a bit of time.");
    await disassociateAddress(associationId);
    await terminateInstance(instanceId);
    await releaseAddress(ipAllocationId);
    await deleteSecurityGroup(securityGroupId);
    deleteTemporaryDirectory();
    await deleteKeyPair(keyPairName);
    console.log(
      "Done cleaning up. Thanks for staying until the end!",
      "If you have any feedback please use the feedback button in the docs",
      "or create an issue on GitHub.",
    );
  }
};
```

- 有关 API 详细信息，请参阅《Amazon SDK for JavaScript API 参考》中的以下主题。
  - AllocateAddress
  - AssociateAddress
  - AuthorizeSecurityGroupIngress
  - CreateKeyPair
  - CreateSecurityGroup
  - DeleteKeyPair
  - DeleteSecurityGroup
  - DescribeImages
  - DescribeInstanceTypes
  - DescribeInstances
  - DescribeKeyPairs
  - DescribeSecurityGroups
  - DisassociateAddress
  - ReleaseAddress
  - RunInstances
  - StartInstances
  - StopInstances
  - TerminateInstances

- UnmonitorInstances

Elastic Load Balancing 示例，使用适用于 JavaScript (v3) 的

以下代码示例向您展示了如何使用带有 Elastic Load Balancing 的 Amazon SDK for JavaScript (v3) 来执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景 是展示如何通过在同一服务中调用多个函数来完成特定任务的代码示例。

每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关如何在上下文中设置和运行代码的说明。

开始使用

开始使用 Elastic Load Balancing

以下代码示例演示了如何开始使用 Elastic Load Balancing。

适用于 JavaScript (v3) 的软件开发工具包

> Note
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

import {
  ElasticLoadBalancingV2Client,
  DescribeLoadBalancersCommand,
} from "@aws-sdk/client-elastic-load-balancing-v2";

export async function main() {
  const client = new ElasticLoadBalancingV2Client({});
  const { LoadBalancers } = await client.send(
    new DescribeLoadBalancersCommand({}),
  );
```

```
  const loadBalancersList = LoadBalancers.map(
    (lb) => `• ${lb.LoadBalancerName}: ${lb.DNSName}`,
  ).join("\n");
  console.log(
    "Hello, Elastic Load Balancing! Let's list some of your load balancers:\n",
    loadBalancersList,
  );
}

// Call function if run directly
import { fileURLToPath } from "url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main();
}
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DescribeLoadBalancers中的。

主题

- 操作
- 场景

操作

为负载均衡器创建侦听器

以下代码示例演示了如何创建一个将请求从 ELB 负载均衡器转发到目标组的侦听器。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    const client = new ElasticLoadBalancingV2Client({});
    const { Listeners } = await client.send(
      new CreateListenerCommand({
```

```
      LoadBalancerArn: state.loadBalancerArn,
      Protocol: state.targetGroupProtocol,
      Port: state.targetGroupPort,
      DefaultActions: [
        { Type: "forward", TargetGroupArn: state.targetGroupArn },
      ],
    }),
  );
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考CreateListener中的。

创建目标组

以下代码示例演示了如何创建 ELB 目标组。

适用于 JavaScript (v3) 的软件开发工具包

Note

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
  const client = new ElasticLoadBalancingV2Client({});
  const { TargetGroups } = await client.send(
    new CreateTargetGroupCommand({
      Name: NAMES.loadBalancerTargetGroupName,
      Protocol: "HTTP",
      Port: 80,
      HealthCheckPath: "/healthcheck",
      HealthCheckIntervalSeconds: 10,
      HealthCheckTimeoutSeconds: 5,
      HealthyThresholdCount: 2,
      UnhealthyThresholdCount: 2,
      VpcId: state.defaultVpc,
    }),
  );
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考CreateTargetGroup中的。

创建应用程序负载均衡器

以下代码示例演示了如何创建 ELB 应用程序负载均衡器。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
const client = new ElasticLoadBalancingV2Client({});
const { LoadBalancers } = await client.send(
  new CreateLoadBalancerCommand({
    Name: NAMES.loadBalancerName,
    Subnets: state.subnets,
  }),
);
state.loadBalancerDns = LoadBalancers[0].DNSName;
state.loadBalancerArn = LoadBalancers[0].LoadBalancerArn;
await waitUntilLoadBalancerAvailable(
  { client },
  { Names: [NAMES.loadBalancerName] },
);
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考CreateLoadBalancer中的。

删除负载均衡器

以下代码示例演示了如何删除 ELB 负载均衡器。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    const client = new ElasticLoadBalancingV2Client({});
    const loadBalancer = await findLoadBalancer(NAMES.loadBalancerName);
    await client.send(
      new DeleteLoadBalancerCommand({
        LoadBalancerArn: loadBalancer.LoadBalancerArn,
      }),
    );
    await retry({ intervalInMs: 1000, maxRetries: 60 }, async () => {
      const lb = await findLoadBalancer(NAMES.loadBalancerName);
      if (lb) {
        throw new Error("Load balancer still exists.");
      }
    });
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DeleteLoadBalancer中的。

删除目标组

以下代码示例演示了如何删除 ELB 目标组。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
  const client = new ElasticLoadBalancingV2Client({});
  try {
    const { TargetGroups } = await client.send(
      new DescribeTargetGroupsCommand({
        Names: [NAMES.loadBalancerTargetGroupName],
      }),
    );

    await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
      client.send(
        new DeleteTargetGroupCommand({
          TargetGroupArn: TargetGroups[0].TargetGroupArn,
```

```
        }),
      ),
    );
  } catch (e) {
    state.deleteLoadBalancerTargetGroupError = e;
  }
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DeleteTargetGroup中的。

描述目标组

以下代码示例演示了如何描述特定目标组。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
const client = new ElasticLoadBalancingV2Client({});
const { TargetGroups } = await client.send(
  new DescribeTargetGroupsCommand({
    Names: [NAMES.loadBalancerTargetGroupName],
  }),
);
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DescribeTargetGroups中的。

获取负载均衡器的端点

以下代码示例演示了如何获取 ELB 负载均衡器的端点。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

import {
  ElasticLoadBalancingV2Client,
  DescribeLoadBalancersCommand,
} from "@aws-sdk/client-elastic-load-balancing-v2";

export async function main() {
  const client = new ElasticLoadBalancingV2Client({});
  const { LoadBalancers } = await client.send(
    new DescribeLoadBalancersCommand({}),
  );
  const loadBalancersList = LoadBalancers.map(
    (lb) => `• ${lb.LoadBalancerName}: ${lb.DNSName}`,
  ).join("\n");
  console.log(
    "Hello, Elastic Load Balancing! Let's list some of your load balancers:\n",
    loadBalancersList,
  );
}

// Call function if run directly
import { fileURLToPath } from "url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main();
}
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DescribeLoadBalancers中的。

获取目标组的运行状况

以下代码示例演示了如何获取 ELB 目标组中实例的运行状况。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
const { TargetHealthDescriptions } = await client.send(
  new DescribeTargetHealthCommand({
    TargetGroupArn: TargetGroups[0].TargetGroupArn,
  }),
);
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DescribeTargetHealth中的。

场景

构建和管理弹性服务

以下代码示例演示了如何创建可返回书籍、电影和歌曲推荐的负载均衡的 Web 服务。该示例演示服务如何响应故障，以及如何重组服务以提高故障发生时的弹性。

- 使用 Amazon EC2 Auto Scaling 组根据启动模板创建 Amazon Elastic Compute Cloud（Amazon EC2）实例，并将实例数量保持在指定范围内。
- 使用弹性负载均衡处理和分发 HTTP 请求。
- 监控自动扩缩组中实例的运行状况，并仅将请求转发到运行状况良好的实例。
- 在每个 EC2 实例上运行 Python Web 服务器以处理 HTTP 请求。Web 服务器以建议和运行状况检查作为响应。
- 使用 Amazon DynamoDB 表模拟推荐服务。
- 通过更新 Amazon Systems Manager 参数来控制 Web 服务器对请求和运行状况检查的响应。

适用于 JavaScript (v3) 的软件开发工具包

Note

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

在命令提示符中运行交互式场景。

```
#!/usr/bin/env node
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0


import {
  Scenario,
  parseScenarioArgs,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

/**
 * The workflow steps are split into three stages:
 *   - deploy
 *   - demo
 *   - destroy
 *
 * Each of these stages has a corresponding file prefixed with steps-*.
 */
import { deploySteps } from "./steps-deploy.js";
import { demoSteps } from "./steps-demo.js";
import { destroySteps } from "./steps-destroy.js";

/**
 * The context is passed to every scenario. Scenario steps
 * will modify the context.
 */
const context = {};

/**
 * Three Scenarios are created for the workflow. A Scenario is an orchestration
 class
 * that simplifies running a series of steps.
 */
```

```
export const scenarios = {
  // Deploys all resources necessary for the workflow.
  deploy: new Scenario("Resilient Workflow - Deploy", deploySteps, context),
  // Demonstrates how a fragile web service can be made more resilient.
  demo: new Scenario("Resilient Workflow - Demo", demoSteps, context),
  // Destroys the resources created for the workflow.
  destroy: new Scenario("Resilient Workflow - Destroy", destroySteps, context),
};

// Call function if run directly
import { fileURLToPath } from "url";

if (process.argv[1] === fileURLToPath(import.meta.url)) {
  parseScenarioArgs(scenarios);
}
```

创建部署所有资源的步骤。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { join } from "node:path";
import { readFileSync, writeFileSync } from "node:fs";
import axios from "axios";

import {
  BatchWriteItemCommand,
  CreateTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";
import {
  EC2Client,
  CreateKeyPairCommand,
  CreateLaunchTemplateCommand,
  DescribeAvailabilityZonesCommand,
  DescribeVpcsCommand,
  DescribeSubnetsCommand,
  DescribeSecurityGroupsCommand,
  AuthorizeSecurityGroupIngressCommand,
} from "@aws-sdk/client-ec2";
import {
  IAMClient,
```

```
  CreatePolicyCommand,
  CreateRoleCommand,
  CreateInstanceProfileCommand,
  AddRoleToInstanceProfileCommand,
  AttachRolePolicyCommand,
  waitUntilInstanceProfileExists,
} from "@aws-sdk/client-iam";
import { SSMClient, GetParameterCommand } from "@aws-sdk/client-ssm";
import {
  CreateAutoScalingGroupCommand,
  AutoScalingClient,
  AttachLoadBalancerTargetGroupsCommand,
} from "@aws-sdk/client-auto-scaling";
import {
  CreateListenerCommand,
  CreateLoadBalancerCommand,
  CreateTargetGroupCommand,
  ElasticLoadBalancingV2Client,
  waitUntilLoadBalancerAvailable,
} from "@aws-sdk/client-elastic-load-balancing-v2";

import {
  ScenarioOutput,
  ScenarioInput,
  ScenarioAction,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES, RESOURCES_PATH, ROOT } from "./constants.js";
import { initParamsSteps } from "./steps-reset-params.js";

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const deploySteps = [
  new ScenarioOutput("introduction", MESSAGES.introduction, { header: true }),
  new ScenarioInput("confirmDeployment", MESSAGES.confirmDeployment, {
    type: "confirm",
  }),
  new ScenarioAction(
    "handleConfirmDeployment",
    (c) => c.confirmDeployment === false && process.exit(),
  ),
  new ScenarioOutput(
```

```
      "creatingTable",
      MESSAGES.creatingTable.replace("${TABLE_NAME}", NAMES.tableName),
    ),
    new ScenarioAction("createTable", async () => {
      const client = new DynamoDBClient({});
      await client.send(
        new CreateTableCommand({
          TableName: NAMES.tableName,
          ProvisionedThroughput: {
            ReadCapacityUnits: 5,
            WriteCapacityUnits: 5,
          },
          AttributeDefinitions: [
            {
              AttributeName: "MediaType",
              AttributeType: "S",
            },
            {
              AttributeName: "ItemId",
              AttributeType: "N",
            },
          ],
          KeySchema: [
            {
              AttributeName: "MediaType",
              KeyType: "HASH",
            },
            {
              AttributeName: "ItemId",
              KeyType: "RANGE",
            },
          ],
        }),
      );
      await waitUntilTableExists({ client }, { TableName: NAMES.tableName });
    }),
    new ScenarioOutput(
      "createdTable",
      MESSAGES.createdTable.replace("${TABLE_NAME}", NAMES.tableName),
    ),
    new ScenarioOutput(
      "populatingTable",
      MESSAGES.populatingTable.replace("${TABLE_NAME}", NAMES.tableName),
    ),
```

```
  new ScenarioAction("populateTable", () => {
    const client = new DynamoDBClient({});
    /**
     * @type {{ default: import("@aws-sdk/client-dynamodb").PutRequest['Item'][] }}
     */
    const recommendations = JSON.parse(
      readFileSync(join(RESOURCES_PATH, "recommendations.json")),
    );

    return client.send(
      new BatchWriteItemCommand({
        RequestItems: {
          [NAMES.tableName]: recommendations.map((item) => ({
            PutRequest: { Item: item },
          })),
        },
      }),
    );
  }),
  new ScenarioOutput(
    "populatedTable",
    MESSAGES.populatedTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
  new ScenarioOutput(
    "creatingKeyPair",
    MESSAGES.creatingKeyPair.replace("${KEY_PAIR_NAME}", NAMES.keyPairName),
  ),
  new ScenarioAction("createKeyPair", async () => {
    const client = new EC2Client({});
    const { KeyMaterial } = await client.send(
      new CreateKeyPairCommand({
        KeyName: NAMES.keyPairName,
      }),
    );

    writeFileSync(`${NAMES.keyPairName}.pem`, KeyMaterial, { mode: 0o600 });
  }),
  new ScenarioOutput(
    "createdKeyPair",
    MESSAGES.createdKeyPair.replace("${KEY_PAIR_NAME}", NAMES.keyPairName),
  ),
  new ScenarioOutput(
    "creatingInstancePolicy",
    MESSAGES.creatingInstancePolicy.replace(
```

```
      "${INSTANCE_POLICY_NAME}",
      NAMES.instancePolicyName,
    ),
  ),
  new ScenarioAction("createInstancePolicy", async (state) => {
    const client = new IAMClient({});
    const {
      Policy: { Arn },
    } = await client.send(
      new CreatePolicyCommand({
        PolicyName: NAMES.instancePolicyName,
        PolicyDocument: readFileSync(
          join(RESOURCES_PATH, "instance_policy.json"),
        ),
      }),
    );
    state.instancePolicyArn = Arn;
  }),
  new ScenarioOutput("createdInstancePolicy", (state) =>
    MESSAGES.createdInstancePolicy
      .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
      .replace("${INSTANCE_POLICY_ARN}", state.instancePolicyArn),
  ),
  new ScenarioOutput(
    "creatingInstanceRole",
    MESSAGES.creatingInstanceRole.replace(
      "${INSTANCE_ROLE_NAME}",
      NAMES.instanceRoleName,
    ),
  ),
  new ScenarioAction("createInstanceRole", () => {
    const client = new IAMClient({});
    return client.send(
      new CreateRoleCommand({
        RoleName: NAMES.instanceRoleName,
        AssumeRolePolicyDocument: readFileSync(
          join(ROOT, "assume-role-policy.json"),
        ),
      }),
    );
  }),
  new ScenarioOutput(
    "createdInstanceRole",
    MESSAGES.createdInstanceRole.replace(
```

```
      "${INSTANCE_ROLE_NAME}",
      NAMES.instanceRoleName,
    ),
  ),
  new ScenarioOutput(
    "attachingPolicyToRole",
    MESSAGES.attachingPolicyToRole
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName)
      .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName),
  ),
  new ScenarioAction("attachPolicyToRole", async (state) => {
    const client = new IAMClient({});
    await client.send(
      new AttachRolePolicyCommand({
        RoleName: NAMES.instanceRoleName,
        PolicyArn: state.instancePolicyArn,
      }),
    );
  }),
  new ScenarioOutput(
    "attachedPolicyToRole",
    MESSAGES.attachedPolicyToRole
      .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
  ),
  new ScenarioOutput(
    "creatingInstanceProfile",
    MESSAGES.creatingInstanceProfile.replace(
      "${INSTANCE_PROFILE_NAME}",
      NAMES.instanceProfileName,
    ),
  ),
  new ScenarioAction("createInstanceProfile", async (state) => {
    const client = new IAMClient({});
    const {
      InstanceProfile: { Arn },
    } = await client.send(
      new CreateInstanceProfileCommand({
        InstanceProfileName: NAMES.instanceProfileName,
      }),
    );
    state.instanceProfileArn = Arn;

    await waitUntilInstanceProfileExists(
```

```
      { client },
      { InstanceProfileName: NAMES.instanceProfileName },
    );
  }),
  new ScenarioOutput("createdInstanceProfile", (state) =>
    MESSAGES.createdInstanceProfile
      .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
      .replace("${INSTANCE_PROFILE_ARN}", state.instanceProfileArn),
  ),
  new ScenarioOutput(
    "addingRoleToInstanceProfile",
    MESSAGES.addingRoleToInstanceProfile
      .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
  ),
  new ScenarioAction("addRoleToInstanceProfile", () => {
    const client = new IAMClient({});
    return client.send(
      new AddRoleToInstanceProfileCommand({
        RoleName: NAMES.instanceRoleName,
        InstanceProfileName: NAMES.instanceProfileName,
      }),
    );
  }),
  new ScenarioOutput(
    "addedRoleToInstanceProfile",
    MESSAGES.addedRoleToInstanceProfile
      .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
  ),
  ...initParamsSteps,
  new ScenarioOutput("creatingLaunchTemplate", MESSAGES.creatingLaunchTemplate),
  new ScenarioAction("createLaunchTemplate", async () => {
    // snippet-start:[javascript.v3.wkflw.resilient.CreateLaunchTemplate]
    const ssmClient = new SSMClient({});
    const { Parameter } = await ssmClient.send(
      new GetParameterCommand({
        Name: "/aws/service/ami-amazon-linux-latest/amzn2-ami-hvm-x86_64-gp2",
      }),
    );
    const ec2Client = new EC2Client({});
    await ec2Client.send(
      new CreateLaunchTemplateCommand({
        LaunchTemplateName: NAMES.launchTemplateName,
```

```
        LaunchTemplateData: {
          InstanceType: "t3.micro",
          ImageId: Parameter.Value,
          IamInstanceProfile: { Name: NAMES.instanceProfileName },
          UserData: readFileSync(
            join(RESOURCES_PATH, "server_startup_script.sh"),
          ).toString("base64"),
          KeyName: NAMES.keyPairName,
        },
      }),
      // snippet-end:[javascript.v3.wkflw.resilient.CreateLaunchTemplate]
    );
  }),
  new ScenarioOutput(
    "createdLaunchTemplate",
    MESSAGES.createdLaunchTemplate.replace(
      "${LAUNCH_TEMPLATE_NAME}",
      NAMES.launchTemplateName,
    ),
  ),
  new ScenarioOutput(
    "creatingAutoScalingGroup",
    MESSAGES.creatingAutoScalingGroup.replace(
      "${AUTO_SCALING_GROUP_NAME}",
      NAMES.autoScalingGroupName,
    ),
  ),
  new ScenarioAction("createAutoScalingGroup", async (state) => {
    const ec2Client = new EC2Client({});
    const { AvailabilityZones } = await ec2Client.send(
      new DescribeAvailabilityZonesCommand({}),
    );
    state.availabilityZoneNames = AvailabilityZones.map((az) => az.ZoneName);
    const autoScalingClient = new AutoScalingClient({});
    await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
      autoScalingClient.send(
        new CreateAutoScalingGroupCommand({
          AvailabilityZones: state.availabilityZoneNames,
          AutoScalingGroupName: NAMES.autoScalingGroupName,
          LaunchTemplate: {
            LaunchTemplateName: NAMES.launchTemplateName,
            Version: "$Default",
          },
          MinSize: 3,
```

```
          MaxSize: 3,
        }),
      ),
    );
  }),
  new ScenarioOutput(
    "createdAutoScalingGroup",
    /**
     * @param {{ availabilityZoneNames: string[] }} state
     */
    (state) =>
      MESSAGES.createdAutoScalingGroup
        .replace("${AUTO_SCALING_GROUP_NAME}", NAMES.autoScalingGroupName)
        .replace(
          "${AVAILABILITY_ZONE_NAMES}",
          state.availabilityZoneNames.join(", "),
        ),
  ),
  new ScenarioInput("confirmContinue", MESSAGES.confirmContinue, {
    type: "confirm",
  }),
  new ScenarioOutput("loadBalancer", MESSAGES.loadBalancer),
  new ScenarioOutput("gettingVpc", MESSAGES.gettingVpc),
  new ScenarioAction("getVpc", async (state) => {
    // snippet-start:[javascript.v3.wkflw.resilient.DescribeVpcs]
    const client = new EC2Client({});
    const { Vpcs } = await client.send(
      new DescribeVpcsCommand({
        Filters: [{ Name: "is-default", Values: ["true"] }],
      }),
    );
    // snippet-end:[javascript.v3.wkflw.resilient.DescribeVpcs]
    state.defaultVpc = Vpcs[0].VpcId;
  }),
  new ScenarioOutput("gotVpc", (state) =>
    MESSAGES.gotVpc.replace("${VPC_ID}", state.defaultVpc),
  ),
  new ScenarioOutput("gettingSubnets", MESSAGES.gettingSubnets),
  new ScenarioAction("getSubnets", async (state) => {
    // snippet-start:[javascript.v3.wkflw.resilient.DescribeSubnets]
    const client = new EC2Client({});
    const { Subnets } = await client.send(
      new DescribeSubnetsCommand({
        Filters: [
```

```
        { Name: "vpc-id", Values: [state.defaultVpc] },
        { Name: "availability-zone", Values: state.availabilityZoneNames },
        { Name: "default-for-az", Values: ["true"] },
      ],
    }),
  );
  // snippet-end:[javascript.v3.wkflw.resilient.DescribeSubnets]
  state.subnets = Subnets.map((subnet) => subnet.SubnetId);
}),
new ScenarioOutput(
  "gotSubnets",
  /**
   * @param {{ subnets: string[] }} state
   */
  (state) =>
    MESSAGES.gotSubnets.replace("${SUBNETS}", state.subnets.join(", ")),
),
new ScenarioOutput(
  "creatingLoadBalancerTargetGroup",
  MESSAGES.creatingLoadBalancerTargetGroup.replace(
    "${TARGET_GROUP_NAME}",
    NAMES.loadBalancerTargetGroupName,
  ),
),
new ScenarioAction("createLoadBalancerTargetGroup", async (state) => {
  // snippet-start:[javascript.v3.wkflw.resilient.CreateTargetGroup]
  const client = new ElasticLoadBalancingV2Client({});
  const { TargetGroups } = await client.send(
    new CreateTargetGroupCommand({
      Name: NAMES.loadBalancerTargetGroupName,
      Protocol: "HTTP",
      Port: 80,
      HealthCheckPath: "/healthcheck",
      HealthCheckIntervalSeconds: 10,
      HealthCheckTimeoutSeconds: 5,
      HealthyThresholdCount: 2,
      UnhealthyThresholdCount: 2,
      VpcId: state.defaultVpc,
    }),
  );
  // snippet-end:[javascript.v3.wkflw.resilient.CreateTargetGroup]
  const targetGroup = TargetGroups[0];
  state.targetGroupArn = targetGroup.TargetGroupArn;
  state.targetGroupProtocol = targetGroup.Protocol;
```

```
      state.targetGroupPort = targetGroup.Port;
    }),
    new ScenarioOutput(
      "createdLoadBalancerTargetGroup",
      MESSAGES.createdLoadBalancerTargetGroup.replace(
        "${TARGET_GROUP_NAME}",
        NAMES.loadBalancerTargetGroupName,
      ),
    ),
    new ScenarioOutput(
      "creatingLoadBalancer",
      MESSAGES.creatingLoadBalancer.replace("${LB_NAME}", NAMES.loadBalancerName),
    ),
    new ScenarioAction("createLoadBalancer", async (state) => {
      // snippet-start:[javascript.v3.wkflw.resilient.CreateLoadBalancer]
      const client = new ElasticLoadBalancingV2Client({});
      const { LoadBalancers } = await client.send(
        new CreateLoadBalancerCommand({
          Name: NAMES.loadBalancerName,
          Subnets: state.subnets,
        }),
      );
      state.loadBalancerDns = LoadBalancers[0].DNSName;
      state.loadBalancerArn = LoadBalancers[0].LoadBalancerArn;
      await waitUntilLoadBalancerAvailable(
        { client },
        { Names: [NAMES.loadBalancerName] },
      );
      // snippet-end:[javascript.v3.wkflw.resilient.CreateLoadBalancer]
    }),
    new ScenarioOutput("createdLoadBalancer", (state) =>
      MESSAGES.createdLoadBalancer
        .replace("${LB_NAME}", NAMES.loadBalancerName)
        .replace("${DNS_NAME}", state.loadBalancerDns),
    ),
    new ScenarioOutput(
      "creatingListener",
      MESSAGES.creatingLoadBalancerListener
        .replace("${LB_NAME}", NAMES.loadBalancerName)
        .replace("${TARGET_GROUP_NAME}", NAMES.loadBalancerTargetGroupName),
    ),
    new ScenarioAction("createListener", async (state) => {
      // snippet-start:[javascript.v3.wkflw.resilient.CreateListener]
      const client = new ElasticLoadBalancingV2Client({});
```

```
      const { Listeners } = await client.send(
        new CreateListenerCommand({
          LoadBalancerArn: state.loadBalancerArn,
          Protocol: state.targetGroupProtocol,
          Port: state.targetGroupPort,
          DefaultActions: [
            { Type: "forward", TargetGroupArn: state.targetGroupArn },
          ],
        }),
      );
      // snippet-end:[javascript.v3.wkflw.resilient.CreateListener]
      const listener = Listeners[0];
      state.loadBalancerListenerArn = listener.ListenerArn;
    }),
    new ScenarioOutput("createdListener", (state) =>
      MESSAGES.createdLoadBalancerListener.replace(
        "${LB_LISTENER_ARN}",
        state.loadBalancerListenerArn,
      ),
    ),
    new ScenarioOutput(
      "attachingLoadBalancerTargetGroup",
      MESSAGES.attachingLoadBalancerTargetGroup
        .replace("${TARGET_GROUP_NAME}", NAMES.loadBalancerTargetGroupName)
        .replace("${AUTO_SCALING_GROUP_NAME}", NAMES.autoScalingGroupName),
    ),
    new ScenarioAction("attachLoadBalancerTargetGroup", async (state) => {
      // snippet-start:[javascript.v3.wkflw.resilient.AttachTargetGroup]
      const client = new AutoScalingClient({});
      await client.send(
        new AttachLoadBalancerTargetGroupsCommand({
          AutoScalingGroupName: NAMES.autoScalingGroupName,
          TargetGroupARNs: [state.targetGroupArn],
        }),
      );
      // snippet-end:[javascript.v3.wkflw.resilient.AttachTargetGroup]
    }),
    new ScenarioOutput(
      "attachedLoadBalancerTargetGroup",
      MESSAGES.attachedLoadBalancerTargetGroup,
    ),
    new ScenarioOutput("verifyingInboundPort", MESSAGES.verifyingInboundPort),
    new ScenarioAction(
      "verifyInboundPort",
```

```
    /**
     *
     * @param {{ defaultSecurityGroup: import('@aws-sdk/client-ec2').SecurityGroup}}
  state
     */
    async (state) => {
      const client = new EC2Client({});
      const { SecurityGroups } = await client.send(
        new DescribeSecurityGroupsCommand({
          Filters: [{ Name: "group-name", Values: ["default"] }],
        }),
      );
      if (!SecurityGroups) {
        state.verifyInboundPortError = new Error(MESSAGES.noSecurityGroups);
      }
      state.defaultSecurityGroup = SecurityGroups[0];

      /**
       * @type {string}
       */
      const ipResponse = (await axios.get("http://checkip.amazonaws.com")).data;
      state.myIp = ipResponse.trim();
      const myIpRules = state.defaultSecurityGroup.IpPermissions.filter(
        ({ IpRanges }) =>
          IpRanges.some(
            ({ CidrIp }) =>
              CidrIp.startsWith(state.myIp) || CidrIp === "0.0.0.0/0",
          ),
      )
        .filter(({ IpProtocol }) => IpProtocol === "tcp")
        .filter(({ FromPort }) => FromPort === 80);

      state.myIpRules = myIpRules;
    },
  ),
  new ScenarioOutput(
    "verifiedInboundPort",
    /**
     * @param {{ myIpRules: any[] }} state
     */
    (state) => {
      if (state.myIpRules.length > 0) {
        return MESSAGES.foundIpRules.replace(
          "${IP_RULES}",
```

```
          JSON.stringify(state.myIpRules, null, 2),
        );
      } else {
        return MESSAGES.noIpRules;
      }
    },
  ),
  new ScenarioInput(
    "shouldAddInboundRule",
    /**
     * @param {{ myIpRules: any[] }} state
     */
    (state) => {
      if (state.myIpRules.length > 0) {
        return false;
      } else {
        return MESSAGES.noIpRules;
      }
    },
    { type: "confirm" },
  ),
  new ScenarioAction(
    "addInboundRule",
    /**
     * @param {{ defaultSecurityGroup: import('@aws-sdk/client-
ec2').SecurityGroup }} state
     */
    async (state) => {
      if (!state.shouldAddInboundRule) {
        return;
      }

      const client = new EC2Client({});
      await client.send(
        new AuthorizeSecurityGroupIngressCommand({
          GroupId: state.defaultSecurityGroup.GroupId,
          CidrIp: `${state.myIp}/32`,
          FromPort: 80,
          ToPort: 80,
          IpProtocol: "tcp",
        }),
      );
    },
  ),
```

```
  new ScenarioOutput("addedInboundRule", (state) => {
    if (state.shouldAddInboundRule) {
      return MESSAGES.addedInboundRule.replace("${IP_ADDRESS}", state.myIp);
    } else {
      return false;
    }
  }),
  new ScenarioOutput("verifyingEndpoint", (state) =>
    MESSAGES.verifyingEndpoint.replace("${DNS_NAME}", state.loadBalancerDns),
  ),
  new ScenarioAction("verifyEndpoint", async (state) => {
    try {
      const response = await retry({ intervalInMs: 2000, maxRetries: 30 }, () =>
        axios.get(`http://${state.loadBalancerDns}`),
      );
      state.endpointResponse = JSON.stringify(response.data, null, 2);
    } catch (e) {
      state.verifyEndpointError = e;
    }
  }),
  new ScenarioOutput("verifiedEndpoint", (state) => {
    if (state.verifyEndpointError) {
      console.error(state.verifyEndpointError);
    } else {
      return MESSAGES.verifiedEndpoint.replace(
        "${ENDPOINT_RESPONSE}",
        state.endpointResponse,
      );
    }
  }),
];
```

创建运行演示的步骤。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { readFileSync } from "node:fs";
import { join } from "node:path";

import axios from "axios";

import {
```

```
  DescribeTargetGroupsCommand,
  DescribeTargetHealthCommand,
  ElasticLoadBalancingV2Client,
} from "@aws-sdk/client-elastic-load-balancing-v2";
import {
  DescribeInstanceInformationCommand,
  PutParameterCommand,
  SSMClient,
  SendCommandCommand,
} from "@aws-sdk/client-ssm";
import {
  IAMClient,
  CreatePolicyCommand,
  CreateRoleCommand,
  AttachRolePolicyCommand,
  CreateInstanceProfileCommand,
  AddRoleToInstanceProfileCommand,
  waitUntilInstanceProfileExists,
} from "@aws-sdk/client-iam";
import {
  AutoScalingClient,
  DescribeAutoScalingGroupsCommand,
  TerminateInstanceInAutoScalingGroupCommand,
} from "@aws-sdk/client-auto-scaling";
import {
  DescribeIamInstanceProfileAssociationsCommand,
  EC2Client,
  RebootInstancesCommand,
  ReplaceIamInstanceProfileAssociationCommand,
} from "@aws-sdk/client-ec2";

import {
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/scenario.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES, RESOURCES_PATH } from "./constants.js";
import { findLoadBalancer } from "./shared.js";

const getRecommendation = new ScenarioAction(
  "getRecommendation",
  async (state) => {
```

```
    const loadBalancer = await findLoadBalancer(NAMES.loadBalancerName);
    if (loadBalancer) {
      state.loadBalancerDnsName = loadBalancer.DNSName;
      try {
        state.recommendation = (
          await axios.get(`http://${state.loadBalancerDnsName}`)
        ).data;
      } catch (e) {
        state.recommendation = e instanceof Error ? e.message : e;
      }
    } else {
      throw new Error(MESSAGES.demoFindLoadBalancerError);
    }
  },
);

const getRecommendationResult = new ScenarioOutput(
  "getRecommendationResult",
  (state) =>
    `Recommendation:\n${JSON.stringify(state.recommendation, null, 2)}`,
  { preformatted: true },
);

const getHealthCheck = new ScenarioAction("getHealthCheck", async (state) => {
  // snippet-start:[javascript.v3.wkflw.resilient.DescribeTargetGroups]
  const client = new ElasticLoadBalancingV2Client({});
  const { TargetGroups } = await client.send(
    new DescribeTargetGroupsCommand({
      Names: [NAMES.loadBalancerTargetGroupName],
    }),
  );
  // snippet-end:[javascript.v3.wkflw.resilient.DescribeTargetGroups]

  // snippet-start:[javascript.v3.wkflw.resilient.DescribeTargetHealth]
  const { TargetHealthDescriptions } = await client.send(
    new DescribeTargetHealthCommand({
      TargetGroupArn: TargetGroups[0].TargetGroupArn,
    }),
  );
  // snippet-end:[javascript.v3.wkflw.resilient.DescribeTargetHealth]
  state.targetHealthDescriptions = TargetHealthDescriptions;
});

const getHealthCheckResult = new ScenarioOutput(
```

```
  "getHealthCheckResult",
  /**
   * @param {{ targetHealthDescriptions: import('@aws-sdk/client-elastic-load-
balancing-v2').TargetHealthDescription[]}} state
   */
  (state) => {
    const status = state.targetHealthDescriptions
      .map((th) => `${th.Target.Id}: ${th.TargetHealth.State}`)
      .join("\n");
    return `Health check:\n${status}`;
  },
  { preformatted: true },
);

const loadBalancerLoop = new ScenarioAction(
  "loadBalancerLoop",
  getRecommendation.action,
  {
    whileConfig: {
      inputEquals: true,
      input: new ScenarioInput(
        "loadBalancerCheck",
        MESSAGES.demoLoadBalancerCheck,
        {
          type: "confirm",
        },
      ),
      output: getRecommendationResult,
    },
  },
);

const healthCheckLoop = new ScenarioAction(
  "healthCheckLoop",
  getHealthCheck.action,
  {
    whileConfig: {
      inputEquals: true,
      input: new ScenarioInput("healthCheck", MESSAGES.demoHealthCheck, {
        type: "confirm",
      }),
      output: getHealthCheckResult,
    },
  },
```

```
);

const statusSteps = [
  getRecommendation,
  getRecommendationResult,
  getHealthCheck,
  getHealthCheckResult,
];

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const demoSteps = [
  new ScenarioOutput("header", MESSAGES.demoHeader, { header: true }),
  new ScenarioOutput("sanityCheck", MESSAGES.demoSanityCheck),
  ...statusSteps,
  new ScenarioInput(
    "brokenDependencyConfirmation",
    MESSAGES.demoBrokenDependencyConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("brokenDependency", async (state) => {
    if (!state.brokenDependencyConfirmation) {
      process.exit();
    } else {
      const client = new SSMClient({});
      state.badTableName = `fake-table-${Date.now()}`;
      await client.send(
        new PutParameterCommand({
          Name: NAMES.ssmTableNameKey,
          Value: state.badTableName,
          Overwrite: true,
          Type: "String",
        }),
      );
    }
  }),
  new ScenarioOutput("testBrokenDependency", (state) =>
    MESSAGES.demoTestBrokenDependency.replace(
      "${TABLE_NAME}",
      state.badTableName,
    ),
  ),
  ...statusSteps,
```

```
  new ScenarioInput(
    "staticResponseConfirmation",
    MESSAGES.demoStaticResponseConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("staticResponse", async (state) => {
    if (!state.staticResponseConfirmation) {
      process.exit();
    } else {
      const client = new SSMClient({});
      await client.send(
        new PutParameterCommand({
          Name: NAMES.ssmFailureResponseKey,
          Value: "static",
          Overwrite: true,
          Type: "String",
        }),
      );
    }
  }),
  new ScenarioOutput("testStaticResponse", MESSAGES.demoTestStaticResponse),
  ...statusSteps,
  new ScenarioInput(
    "badCredentialsConfirmation",
    MESSAGES.demoBadCredentialsConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("badCredentialsExit", (state) => {
    if (!state.badCredentialsConfirmation) {
      process.exit();
    }
  }),
  new ScenarioAction("fixDynamoDBName", async () => {
    const client = new SSMClient({});
    await client.send(
      new PutParameterCommand({
        Name: NAMES.ssmTableNameKey,
        Value: NAMES.tableName,
        Overwrite: true,
        Type: "String",
      }),
    );
  }),
  new ScenarioAction(
```

```
    "badCredentials",
    /**
     * @param {{ targetInstance: import('@aws-sdk/client-auto-scaling').Instance }}
 state
     */
    async (state) => {
      await createSsmOnlyInstanceProfile();
      const autoScalingClient = new AutoScalingClient({});
      const { AutoScalingGroups } = await autoScalingClient.send(
        new DescribeAutoScalingGroupsCommand({
          AutoScalingGroupNames: [NAMES.autoScalingGroupName],
        }),
      );
      state.targetInstance = AutoScalingGroups[0].Instances[0];
      // snippet-start:
[javascript.v3.wkflw.resilient.DescribeIamInstanceProfileAssociations]
      const ec2Client = new EC2Client({});
      const { IamInstanceProfileAssociations } = await ec2Client.send(
        new DescribeIamInstanceProfileAssociationsCommand({
          Filters: [
            { Name: "instance-id", Values: [state.targetInstance.InstanceId] },
          ],
        }),
      );
      // snippet-end:
[javascript.v3.wkflw.resilient.DescribeIamInstanceProfileAssociations]
      state.instanceProfileAssociationId =
        IamInstanceProfileAssociations[0].AssociationId;
      // snippet-start:
[javascript.v3.wkflw.resilient.ReplaceIamInstanceProfileAssociation]
      await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
        ec2Client.send(
          new ReplaceIamInstanceProfileAssociationCommand({
            AssociationId: state.instanceProfileAssociationId,
            IamInstanceProfile: { Name: NAMES.ssmOnlyInstanceProfileName },
          }),
        ),
      );
      // snippet-end:
[javascript.v3.wkflw.resilient.ReplaceIamInstanceProfileAssociation]

      await ec2Client.send(
        new RebootInstancesCommand({
          InstanceIds: [state.targetInstance.InstanceId],
```

```
        }),
      );

      const ssmClient = new SSMClient({});
      await retry({ intervalInMs: 20000, maxRetries: 15 }, async () => {
        const { InstanceInformationList } = await ssmClient.send(
          new DescribeInstanceInformationCommand({}),
        );

        const instance = InstanceInformationList.find(
          (info) => info.InstanceId === state.targetInstance.InstanceId,
        );

        if (!instance) {
          throw new Error("Instance not found.");
        }
      });

      await ssmClient.send(
        new SendCommandCommand({
          InstanceIds: [state.targetInstance.InstanceId],
          DocumentName: "AWS-RunShellScript",
          Parameters: { commands: ["cd / && sudo python3 server.py 80"] },
        }),
      );
    },
  ),
  new ScenarioOutput(
    "testBadCredentials",
    /**
     * @param {{ targetInstance: import('@aws-sdk/client-ssm').InstanceInformation}}
 state
     */
    (state) =>
      MESSAGES.demoTestBadCredentials.replace(
        "${INSTANCE_ID}",
        state.targetInstance.InstanceId,
      ),
  ),
  loadBalancerLoop,
  new ScenarioInput(
    "deepHealthCheckConfirmation",
    MESSAGES.demoDeepHealthCheckConfirmation,
    { type: "confirm" },
```

```
  ),
  new ScenarioAction("deepHealthCheckExit", (state) => {
    if (!state.deepHealthCheckConfirmation) {
      process.exit();
    }
  }),
  new ScenarioAction("deepHealthCheck", async () => {
    const client = new SSMClient({});
    await client.send(
      new PutParameterCommand({
        Name: NAMES.ssmHealthCheckKey,
        Value: "deep",
        Overwrite: true,
        Type: "String",
      }),
    );
  }),
  new ScenarioOutput("testDeepHealthCheck", MESSAGES.demoTestDeepHealthCheck),
  healthCheckLoop,
  loadBalancerLoop,
  new ScenarioInput(
    "killInstanceConfirmation",
    /**
     * @param {{ targetInstance: import('@aws-sdk/client-
ssm').InstanceInformation }} state
     */
    (state) =>
      MESSAGES.demoKillInstanceConfirmation.replace(
        "${INSTANCE_ID}",
        state.targetInstance.InstanceId,
      ),
    { type: "confirm" },
  ),
  new ScenarioAction("killInstanceExit", (state) => {
    if (!state.killInstanceConfirmation) {
      process.exit();
    }
  }),
  new ScenarioAction(
    "killInstance",
    /**
     * @param {{ targetInstance: import('@aws-sdk/client-
ssm').InstanceInformation }} state
     */
```

```
    async (state) => {
      const client = new AutoScalingClient({});
      await client.send(
        new TerminateInstanceInAutoScalingGroupCommand({
          InstanceId: state.targetInstance.InstanceId,
          ShouldDecrementDesiredCapacity: false,
        }),
      );
    },
  ),
  new ScenarioOutput("testKillInstance", MESSAGES.demoTestKillInstance),
  healthCheckLoop,
  loadBalancerLoop,
  new ScenarioInput("failOpenConfirmation", MESSAGES.demoFailOpenConfirmation, {
    type: "confirm",
  }),
  new ScenarioAction("failOpenExit", (state) => {
    if (!state.failOpenConfirmation) {
      process.exit();
    }
  }),
  new ScenarioAction("failOpen", () => {
    const client = new SSMClient({});
    return client.send(
      new PutParameterCommand({
        Name: NAMES.ssmTableNameKey,
        Value: `fake-table-${Date.now()}`,
        Overwrite: true,
        Type: "String",
      }),
    );
  }),
  new ScenarioOutput("testFailOpen", MESSAGES.demoFailOpenTest),
  healthCheckLoop,
  loadBalancerLoop,
  new ScenarioInput(
    "resetTableConfirmation",
    MESSAGES.demoResetTableConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("resetTableExit", (state) => {
    if (!state.resetTableConfirmation) {
      process.exit();
    }
```

```
  }),
  new ScenarioAction("resetTable", async () => {
    const client = new SSMClient({});
    await client.send(
      new PutParameterCommand({
        Name: NAMES.ssmTableNameKey,
        Value: NAMES.tableName,
        Overwrite: true,
        Type: "String",
      }),
    );
  }),
  new ScenarioOutput("testResetTable", MESSAGES.demoTestResetTable),
  healthCheckLoop,
  loadBalancerLoop,
];

async function createSsmOnlyInstanceProfile() {
  const iamClient = new IAMClient({});
  const { Policy } = await iamClient.send(
    new CreatePolicyCommand({
      PolicyName: NAMES.ssmOnlyPolicyName,
      PolicyDocument: readFileSync(
        join(RESOURCES_PATH, "ssm_only_policy.json"),
      ),
    }),
  );
  await iamClient.send(
    new CreateRoleCommand({
      RoleName: NAMES.ssmOnlyRoleName,
      AssumeRolePolicyDocument: JSON.stringify({
        Version: "2012-10-17",
        Statement: [
          {
            Effect: "Allow",
            Principal: { Service: "ec2.amazonaws.com" },
            Action: "sts:AssumeRole",
          },
        ],
      }),
    }),
  );
  await iamClient.send(
    new AttachRolePolicyCommand({
```

```
      RoleName: NAMES.ssmOnlyRoleName,
      PolicyArn: Policy.Arn,
    }),
  );
  await iamClient.send(
    new AttachRolePolicyCommand({
      RoleName: NAMES.ssmOnlyRoleName,
      PolicyArn: "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore",
    }),
  );
  // snippet-start:[javascript.v3.wkflw.resilient.CreateInstanceProfile]
  const { InstanceProfile } = await iamClient.send(
    new CreateInstanceProfileCommand({
      InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
    }),
  );
  await waitUntilInstanceProfileExists(
    { client: iamClient },
    { InstanceProfileName: NAMES.ssmOnlyInstanceProfileName },
  );
  // snippet-end:[javascript.v3.wkflw.resilient.CreateInstanceProfile]
  await iamClient.send(
    new AddRoleToInstanceProfileCommand({
      InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
      RoleName: NAMES.ssmOnlyRoleName,
    }),
  );

  return InstanceProfile;
}
```

创建销毁所有资源的步骤。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { unlinkSync } from "node:fs";

import { DynamoDBClient, DeleteTableCommand } from "@aws-sdk/client-dynamodb";
import {
  EC2Client,
  DeleteKeyPairCommand,
  DeleteLaunchTemplateCommand,
```

```
} from "@aws-sdk/client-ec2";
import {
  IAMClient,
  DeleteInstanceProfileCommand,
  RemoveRoleFromInstanceProfileCommand,
  DeletePolicyCommand,
  DeleteRoleCommand,
  DetachRolePolicyCommand,
  paginateListPolicies,
} from "@aws-sdk/client-iam";
import {
  AutoScalingClient,
  DeleteAutoScalingGroupCommand,
  TerminateInstanceInAutoScalingGroupCommand,
  UpdateAutoScalingGroupCommand,
  paginateDescribeAutoScalingGroups,
} from "@aws-sdk/client-auto-scaling";
import {
  DeleteLoadBalancerCommand,
  DeleteTargetGroupCommand,
  DescribeTargetGroupsCommand,
  ElasticLoadBalancingV2Client,
} from "@aws-sdk/client-elastic-load-balancing-v2";

import {
  ScenarioOutput,
  ScenarioInput,
  ScenarioAction,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES } from "./constants.js";
import { findLoadBalancer } from "./shared.js";

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const destroySteps = [
  new ScenarioInput("destroy", MESSAGES.destroy, { type: "confirm" }),
  new ScenarioAction(
    "abort",
    (state) => state.destroy === false && process.exit(),
  ),
  new ScenarioAction("deleteTable", async (c) => {
```

```
    try {
      const client = new DynamoDBClient({});
      await client.send(new DeleteTableCommand({ TableName: NAMES.tableName }));
    } catch (e) {
      c.deleteTableError = e;
    }
  }),
  new ScenarioOutput("deleteTableResult", (state) => {
    if (state.deleteTableError) {
      console.error(state.deleteTableError);
      return MESSAGES.deleteTableError.replace(
        "${TABLE_NAME}",
        NAMES.tableName,
      );
    } else {
      return MESSAGES.deletedTable.replace("${TABLE_NAME}", NAMES.tableName);
    }
  }),
  new ScenarioAction("deleteKeyPair", async (state) => {
    try {
      const client = new EC2Client({});
      await client.send(
        new DeleteKeyPairCommand({ KeyName: NAMES.keyPairName }),
      );
      unlinkSync(`${NAMES.keyPairName}.pem`);
    } catch (e) {
      state.deleteKeyPairError = e;
    }
  }),
  new ScenarioOutput("deleteKeyPairResult", (state) => {
    if (state.deleteKeyPairError) {
      console.error(state.deleteKeyPairError);
      return MESSAGES.deleteKeyPairError.replace(
        "${KEY_PAIR_NAME}",
        NAMES.keyPairName,
      );
    } else {
      return MESSAGES.deletedKeyPair.replace(
        "${KEY_PAIR_NAME}",
        NAMES.keyPairName,
      );
    }
  }),
  new ScenarioAction("detachPolicyFromRole", async (state) => {
```

```
    try {
      const client = new IAMClient({});
      const policy = await findPolicy(NAMES.instancePolicyName);

      if (!policy) {
        state.detachPolicyFromRoleError = new Error(
          `Policy ${NAMES.instancePolicyName} not found.`,
        );
      } else {
        await client.send(
          new DetachRolePolicyCommand({
            RoleName: NAMES.instanceRoleName,
            PolicyArn: policy.Arn,
          }),
        );
      }
    } catch (e) {
      state.detachPolicyFromRoleError = e;
    }
  }),
  new ScenarioOutput("detachedPolicyFromRole", (state) => {
    if (state.detachPolicyFromRoleError) {
      console.error(state.detachPolicyFromRoleError);
      return MESSAGES.detachPolicyFromRoleError
        .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    } else {
      return MESSAGES.detachedPolicyFromRole
        .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    }
  }),
  new ScenarioAction("deleteInstancePolicy", async (state) => {
    const client = new IAMClient({});
    const policy = await findPolicy(NAMES.instancePolicyName);

    if (!policy) {
      state.deletePolicyError = new Error(
        `Policy ${NAMES.instancePolicyName} not found.`,
      );
    } else {
      return client.send(
        new DeletePolicyCommand({
          PolicyArn: policy.Arn,
```

```
        }),
      );
    }
  }),
  new ScenarioOutput("deletePolicyResult", (state) => {
    if (state.deletePolicyError) {
      console.error(state.deletePolicyError);
      return MESSAGES.deletePolicyError.replace(
        "${INSTANCE_POLICY_NAME}",
        NAMES.instancePolicyName,
      );
    } else {
      return MESSAGES.deletedPolicy.replace(
        "${INSTANCE_POLICY_NAME}",
        NAMES.instancePolicyName,
      );
    }
  }),
  new ScenarioAction("removeRoleFromInstanceProfile", async (state) => {
    try {
      const client = new IAMClient({});
      await client.send(
        new RemoveRoleFromInstanceProfileCommand({
          RoleName: NAMES.instanceRoleName,
          InstanceProfileName: NAMES.instanceProfileName,
        }),
      );
    } catch (e) {
      state.removeRoleFromInstanceProfileError = e;
    }
  }),
  new ScenarioOutput("removeRoleFromInstanceProfileResult", (state) => {
    if (state.removeRoleFromInstanceProfile) {
      console.error(state.removeRoleFromInstanceProfileError);
      return MESSAGES.removeRoleFromInstanceProfileError
        .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    } else {
      return MESSAGES.removedRoleFromInstanceProfile
        .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    }
  }),
  new ScenarioAction("deleteInstanceRole", async (state) => {
```

```
    try {
      const client = new IAMClient({});
      await client.send(
        new DeleteRoleCommand({
          RoleName: NAMES.instanceRoleName,
        }),
      );
    } catch (e) {
      state.deleteInstanceRoleError = e;
    }
  }),
  new ScenarioOutput("deleteInstanceRoleResult", (state) => {
    if (state.deleteInstanceRoleError) {
      console.error(state.deleteInstanceRoleError);
      return MESSAGES.deleteInstanceRoleError.replace(
        "${INSTANCE_ROLE_NAME}",
        NAMES.instanceRoleName,
      );
    } else {
      return MESSAGES.deletedInstanceRole.replace(
        "${INSTANCE_ROLE_NAME}",
        NAMES.instanceRoleName,
      );
    }
  }),
  new ScenarioAction("deleteInstanceProfile", async (state) => {
    try {
      // snippet-start:[javascript.v3.wkflw.resilient.DeleteInstanceProfile]
      const client = new IAMClient({});
      await client.send(
        new DeleteInstanceProfileCommand({
          InstanceProfileName: NAMES.instanceProfileName,
        }),
      );
      // snippet-end:[javascript.v3.wkflw.resilient.DeleteInstanceProfile]
    } catch (e) {
      state.deleteInstanceProfileError = e;
    }
  }),
  new ScenarioOutput("deleteInstanceProfileResult", (state) => {
    if (state.deleteInstanceProfileError) {
      console.error(state.deleteInstanceProfileError);
      return MESSAGES.deleteInstanceProfileError.replace(
        "${INSTANCE_PROFILE_NAME}",
```

```
        NAMES.instanceProfileName,
      );
    } else {
      return MESSAGES.deletedInstanceProfile.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.instanceProfileName,
      );
    }
  }),
  new ScenarioAction("deleteAutoScalingGroup", async (state) => {
    try {
      await terminateGroupInstances(NAMES.autoScalingGroupName);
      await retry({ intervalInMs: 60000, maxRetries: 60 }, async () => {
        await deleteAutoScalingGroup(NAMES.autoScalingGroupName);
      });
    } catch (e) {
      state.deleteAutoScalingGroupError = e;
    }
  }),
  new ScenarioOutput("deleteAutoScalingGroupResult", (state) => {
    if (state.deleteAutoScalingGroupError) {
      console.error(state.deleteAutoScalingGroupError);
      return MESSAGES.deleteAutoScalingGroupError.replace(
        "${AUTO_SCALING_GROUP_NAME}",
        NAMES.autoScalingGroupName,
      );
    } else {
      return MESSAGES.deletedAutoScalingGroup.replace(
        "${AUTO_SCALING_GROUP_NAME}",
        NAMES.autoScalingGroupName,
      );
    }
  }),
  new ScenarioAction("deleteLaunchTemplate", async (state) => {
    const client = new EC2Client({});
    try {
      // snippet-start:[javascript.v3.wkflw.resilient.DeleteLaunchTemplate]
      await client.send(
        new DeleteLaunchTemplateCommand({
          LaunchTemplateName: NAMES.launchTemplateName,
        }),
      );
      // snippet-end:[javascript.v3.wkflw.resilient.DeleteLaunchTemplate]
    } catch (e) {
```

```
      state.deleteLaunchTemplateError = e;
    }
  }),
  new ScenarioOutput("deleteLaunchTemplateResult", (state) => {
    if (state.deleteLaunchTemplateError) {
      console.error(state.deleteLaunchTemplateError);
      return MESSAGES.deleteLaunchTemplateError.replace(
        "${LAUNCH_TEMPLATE_NAME}",
        NAMES.launchTemplateName,
      );
    } else {
      return MESSAGES.deletedLaunchTemplate.replace(
        "${LAUNCH_TEMPLATE_NAME}",
        NAMES.launchTemplateName,
      );
    }
  }),
  new ScenarioAction("deleteLoadBalancer", async (state) => {
    try {
      // snippet-start:[javascript.v3.wkflw.resilient.DeleteLoadBalancer]
      const client = new ElasticLoadBalancingV2Client({});
      const loadBalancer = await findLoadBalancer(NAMES.loadBalancerName);
      await client.send(
        new DeleteLoadBalancerCommand({
          LoadBalancerArn: loadBalancer.LoadBalancerArn,
        }),
      );
      await retry({ intervalInMs: 1000, maxRetries: 60 }, async () => {
        const lb = await findLoadBalancer(NAMES.loadBalancerName);
        if (lb) {
          throw new Error("Load balancer still exists.");
        }
      });
      // snippet-end:[javascript.v3.wkflw.resilient.DeleteLoadBalancer]
    } catch (e) {
      state.deleteLoadBalancerError = e;
    }
  }),
  new ScenarioOutput("deleteLoadBalancerResult", (state) => {
    if (state.deleteLoadBalancerError) {
      console.error(state.deleteLoadBalancerError);
      return MESSAGES.deleteLoadBalancerError.replace(
        "${LB_NAME}",
        NAMES.loadBalancerName,
```

```
      );
    } else {
      return MESSAGES.deletedLoadBalancer.replace(
        "${LB_NAME}",
        NAMES.loadBalancerName,
      );
    }
  }),
  new ScenarioAction("deleteLoadBalancerTargetGroup", async (state) => {
    // snippet-start:[javascript.v3.wkflw.resilient.DeleteTargetGroup]
    const client = new ElasticLoadBalancingV2Client({});
    try {
      const { TargetGroups } = await client.send(
        new DescribeTargetGroupsCommand({
          Names: [NAMES.loadBalancerTargetGroupName],
        }),
      );

      await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
        client.send(
          new DeleteTargetGroupCommand({
            TargetGroupArn: TargetGroups[0].TargetGroupArn,
          }),
        ),
      );
    } catch (e) {
      state.deleteLoadBalancerTargetGroupError = e;
    }
    // snippet-end:[javascript.v3.wkflw.resilient.DeleteTargetGroup]
  }),
  new ScenarioOutput("deleteLoadBalancerTargetGroupResult", (state) => {
    if (state.deleteLoadBalancerTargetGroupError) {
      console.error(state.deleteLoadBalancerTargetGroupError);
      return MESSAGES.deleteLoadBalancerTargetGroupError.replace(
        "${TARGET_GROUP_NAME}",
        NAMES.loadBalancerTargetGroupName,
      );
    } else {
      return MESSAGES.deletedLoadBalancerTargetGroup.replace(
        "${TARGET_GROUP_NAME}",
        NAMES.loadBalancerTargetGroupName,
      );
    }
  }),
```

```
new ScenarioAction("detachSsmOnlyRoleFromProfile", async (state) => {
  try {
    const client = new IAMClient({});
    await client.send(
      new RemoveRoleFromInstanceProfileCommand({
        InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
        RoleName: NAMES.ssmOnlyRoleName,
      }),
    );
  } catch (e) {
    state.detachSsmOnlyRoleFromProfileError = e;
  }
}),
new ScenarioOutput("detachSsmOnlyRoleFromProfileResult", (state) => {
  if (state.detachSsmOnlyRoleFromProfileError) {
    console.error(state.detachSsmOnlyRoleFromProfileError);
    return MESSAGES.detachSsmOnlyRoleFromProfileError
      .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
      .replace("${PROFILE_NAME}", NAMES.ssmOnlyInstanceProfileName);
  } else {
    return MESSAGES.detachedSsmOnlyRoleFromProfile
      .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
      .replace("${PROFILE_NAME}", NAMES.ssmOnlyInstanceProfileName);
  }
}),
new ScenarioAction("detachSsmOnlyCustomRolePolicy", async (state) => {
  try {
    const iamClient = new IAMClient({});
    const ssmOnlyPolicy = await findPolicy(NAMES.ssmOnlyPolicyName);
    await iamClient.send(
      new DetachRolePolicyCommand({
        RoleName: NAMES.ssmOnlyRoleName,
        PolicyArn: ssmOnlyPolicy.Arn,
      }),
    );
  } catch (e) {
    state.detachSsmOnlyCustomRolePolicyError = e;
  }
}),
new ScenarioOutput("detachSsmOnlyCustomRolePolicyResult", (state) => {
  if (state.detachSsmOnlyCustomRolePolicyError) {
    console.error(state.detachSsmOnlyCustomRolePolicyError);
    return MESSAGES.detachSsmOnlyCustomRolePolicyError
      .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
```

```
        .replace("${POLICY_NAME}", NAMES.ssmOnlyPolicyName);
    } else {
      return MESSAGES.detachedSsmOnlyCustomRolePolicy
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${POLICY_NAME}", NAMES.ssmOnlyPolicyName);
    }
  }),
  new ScenarioAction("detachSsmOnlyAWSRolePolicy", async (state) => {
    try {
      const iamClient = new IAMClient({});
      await iamClient.send(
        new DetachRolePolicyCommand({
          RoleName: NAMES.ssmOnlyRoleName,
          PolicyArn: "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore",
        }),
      );
    } catch (e) {
      state.detachSsmOnlyAWSRolePolicyError = e;
    }
  }),
  new ScenarioOutput("detachSsmOnlyAWSRolePolicyResult", (state) => {
    if (state.detachSsmOnlyAWSRolePolicyError) {
      console.error(state.detachSsmOnlyAWSRolePolicyError);
      return MESSAGES.detachSsmOnlyAWSRolePolicyError
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${POLICY_NAME}", "AmazonSSMManagedInstanceCore");
    } else {
      return MESSAGES.detachedSsmOnlyAWSRolePolicy
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${POLICY_NAME}", "AmazonSSMManagedInstanceCore");
    }
  }),
  new ScenarioAction("deleteSsmOnlyInstanceProfile", async (state) => {
    try {
      const iamClient = new IAMClient({});
      await iamClient.send(
        new DeleteInstanceProfileCommand({
          InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
        }),
      );
    } catch (e) {
      state.deleteSsmOnlyInstanceProfileError = e;
    }
  }),
```

```
  new ScenarioOutput("deleteSsmOnlyInstanceProfileResult", (state) => {
    if (state.deleteSsmOnlyInstanceProfileError) {
      console.error(state.deleteSsmOnlyInstanceProfileError);
      return MESSAGES.deleteSsmOnlyInstanceProfileError.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.ssmOnlyInstanceProfileName,
      );
    } else {
      return MESSAGES.deletedSsmOnlyInstanceProfile.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.ssmOnlyInstanceProfileName,
      );
    }
  }),
  new ScenarioAction("deleteSsmOnlyPolicy", async (state) => {
    try {
      const iamClient = new IAMClient({});
      const ssmOnlyPolicy = await findPolicy(NAMES.ssmOnlyPolicyName);
      await iamClient.send(
        new DeletePolicyCommand({
          PolicyArn: ssmOnlyPolicy.Arn,
        }),
      );
    } catch (e) {
      state.deleteSsmOnlyPolicyError = e;
    }
  }),
  new ScenarioOutput("deleteSsmOnlyPolicyResult", (state) => {
    if (state.deleteSsmOnlyPolicyError) {
      console.error(state.deleteSsmOnlyPolicyError);
      return MESSAGES.deleteSsmOnlyPolicyError.replace(
        "${POLICY_NAME}",
        NAMES.ssmOnlyPolicyName,
      );
    } else {
      return MESSAGES.deletedSsmOnlyPolicy.replace(
        "${POLICY_NAME}",
        NAMES.ssmOnlyPolicyName,
      );
    }
  }),
  new ScenarioAction("deleteSsmOnlyRole", async (state) => {
    try {
      const iamClient = new IAMClient({});
```

```
      await iamClient.send(
        new DeleteRoleCommand({
          RoleName: NAMES.ssmOnlyRoleName,
        }),
      );
    } catch (e) {
      state.deleteSsmOnlyRoleError = e;
    }
  }),
  new ScenarioOutput("deleteSsmOnlyRoleResult", (state) => {
    if (state.deleteSsmOnlyRoleError) {
      console.error(state.deleteSsmOnlyRoleError);
      return MESSAGES.deleteSsmOnlyRoleError.replace(
        "${ROLE_NAME}",
        NAMES.ssmOnlyRoleName,
      );
    } else {
      return MESSAGES.deletedSsmOnlyRole.replace(
        "${ROLE_NAME}",
        NAMES.ssmOnlyRoleName,
      );
    }
  }),
];

/**
 * @param {string} policyName
 */
async function findPolicy(policyName) {
  const client = new IAMClient({});
  const paginatedPolicies = paginateListPolicies({ client }, {});
  for await (const page of paginatedPolicies) {
    const policy = page.Policies.find((p) => p.PolicyName === policyName);
    if (policy) {
      return policy;
    }
  }
}

/**
 * @param {string} groupName
 */
async function deleteAutoScalingGroup(groupName) {
  const client = new AutoScalingClient({});
```

```
  try {
    await client.send(
      new DeleteAutoScalingGroupCommand({
        AutoScalingGroupName: groupName,
      }),
    );
  } catch (err) {
    if (!(err instanceof Error)) {
      throw err;
    } else {
      console.log(err.name);
      throw err;
    }
  }
}

/**
 * @param {string} groupName
 */
async function terminateGroupInstances(groupName) {
  const autoScalingClient = new AutoScalingClient({});
  const group = await findAutoScalingGroup(groupName);
  await autoScalingClient.send(
    new UpdateAutoScalingGroupCommand({
      AutoScalingGroupName: group.AutoScalingGroupName,
      MinSize: 0,
    }),
  );
  for (const i of group.Instances) {
    await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
      autoScalingClient.send(
        new TerminateInstanceInAutoScalingGroupCommand({
          InstanceId: i.InstanceId,
          ShouldDecrementDesiredCapacity: true,
        }),
      ),
    );
  }
}

async function findAutoScalingGroup(groupName) {
  const client = new AutoScalingClient({});
  const paginatedGroups = paginateDescribeAutoScalingGroups({ client }, {});
  for await (const page of paginatedGroups) {
```

```
    const group = page.AutoScalingGroups.find(
      (g) => g.AutoScalingGroupName === groupName,
    );
    if (group) {
      return group;
    }
  }
  throw new Error(`Auto scaling group ${groupName} not found.`);
}
```

- 有关 API 详细信息，请参阅《Amazon SDK for JavaScript API 参考》中的以下主题。
  - AttachLoadBalancerTargetGroups
  - CreateAutoScalingGroup
  - CreateInstanceProfile
  - CreateLaunchTemplate
  - CreateListener
  - CreateLoadBalancer
  - CreateTargetGroup
  - DeleteAutoScalingGroup
  - DeleteInstanceProfile
  - DeleteLaunchTemplate
  - DeleteLoadBalancer
  - DeleteTargetGroup
  - DescribeAutoScalingGroups
  - DescribeAvailabilityZones
  - DescribeIamInstanceProfileAssociations
  - DescribeInstances
  - DescribeLoadBalancers
  - DescribeSubnets
  - DescribeTargetGroups
  - DescribeTargetHealth
  - DescribeVpcs

  - RebootInstances

- ReplaceIamInstanceProfileAssociation
- TerminateInstanceInAutoScalingGroup
- UpdateAutoScalingGroup

EventBridge 使用适用于 JavaScript (v3) 的 SDK 的示例

以下代码示例向您展示了如何通过使用 Amazon SDK for JavaScript (v3) 来执行操作和实现常见场景 EventBridge。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景 是展示如何通过在同一服务中调用多个函数来完成特定任务的代码示例。

每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关如何在上下文中设置和运行代码的说明。

主题

- 操作

操作

添加目标

以下代码示例显示了如何向 Amazon EventBridge 事件添加目标。

适用于 JavaScript (v3) 的软件开发工具包

Note

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

导入 SDK 和客户端模块，然后调用 API。

```
import {
```

```
  EventBridgeClient,
  PutTargetsCommand,
} from "@aws-sdk/client-eventbridge";

export const putTarget = async (
  existingRuleName = "some-rule",
  targetArn = "arn:aws:lambda:us-east-1:000000000000:function:test-func",
  uniqueId = Date.now().toString(),
) => {
  const client = new EventBridgeClient({});
  const response = await client.send(
    new PutTargetsCommand({
      Rule: existingRuleName,
      Targets: [
        {
          Arn: targetArn,
          Id: uniqueId,
        },
      ],
    }),
  );

  console.log("PutTargets response:");
  console.log(response);
  // PutTargets response:
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'f5b23b9a-2c17-45c1-ad5c-f926c3692e3d',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   FailedEntries: [],
  //   FailedEntryCount: 0
  // }

  return response;
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考PutTargets中的。

适用于 JavaScript (v2) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatchEvents service object
var ebevents = new AWS.EventBridge({ apiVersion: "2015-10-07" });

var params = {
  Rule: "DEMO_EVENT",
  Targets: [
    {
      Arn: "LAMBDA_FUNCTION_ARN",
      Id: "myEventBridgeTarget",
    },
  ],
};

ebevents.putTargets(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考PutTargets中的。

创建规则

以下代码示例显示了如何创建 Amazon EventBridge 规则。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

导入 SDK 和客户端模块，然后调用 API。

```
import { EventBridgeClient, PutRuleCommand } from "@aws-sdk/client-eventbridge";

export const putRule = async (
  ruleName = "some-rule",
  source = "some-source",
) => {
  const client = new EventBridgeClient({});

  const response = await client.send(
    new PutRuleCommand({
      Name: ruleName,
      EventPattern: JSON.stringify({ source: [source] }),
      State: "ENABLED",
      EventBusName: "default",
    }),
  );

  console.log("PutRule response:");
  console.log(response);
  // PutRule response:
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'd7292ced-1544-421b-842f-596326bc7072',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   RuleArn: 'arn:aws:events:us-east-1:xxxxxxxxxxxx:rule/
EventBridgeTestRule-1696280037720'
  // }
  return response;
```

```
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考PutRule中的。

适用于 JavaScript (v2) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatchEvents service object
var ebevents = new AWS.EventBridge({ apiVersion: "2015-10-07" });

var params = {
  Name: "DEMO_EVENT",
  RoleArn: "IAM_ROLE_ARN",
  ScheduleExpression: "rate(5 minutes)",
  State: "ENABLED",
};

ebevents.putRule(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.RuleArn);
  }
});
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考PutRule中的。

发送事件

以下代码示例显示了如何发送 Amazon EventBridge 事件。

适用于 JavaScript (v3) 的软件开发工具包

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

导入 SDK 和客户端模块，然后调用 API。

```
import {
  EventBridgeClient,
  PutEventsCommand,
} from "@aws-sdk/client-eventbridge";

export const putEvents = async (
  source = "eventbridge.integration.test",
  detailType = "greeting",
  resources = [],
) => {
  const client = new EventBridgeClient({});

  const response = await client.send(
    new PutEventsCommand({
      Entries: [
        {
          Detail: JSON.stringify({ greeting: "Hello there." }),
          DetailType: detailType,
          Resources: resources,
          Source: source,
        },
      ],
    }),
  );

  console.log("PutEvents response:");
  console.log(response);
  // PutEvents response:
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '3d0df73d-dcea-4a23-ae0d-f5556a3ac109',
  //     extendedRequestId: undefined,
```

```
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   Entries: [ { EventId: '51620841-5af4-6402-d9bc-b77734991eb5' } ],
  //   FailedEntryCount: 0
  // }

  return response;
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考PutEvents中的。

适用于 JavaScript (v2) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatchEvents service object
var ebevents = new AWS.EventBridge({ apiVersion: "2015-10-07" });

var params = {
  Entries: [
    {
      Detail: '{ "key1": "value1", "key2": "value2" }',
      DetailType: "appRequestSubmitted",
      Resources: ["RESOURCE_ARN"],
      Source: "com.company.app",
    },
  ],
};

ebevents.putEvents(params, function (err, data) {
  if (err) {
```

```
    console.log("Error", err);
  } else {
    console.log("Success", data.Entries);
  }
});
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考PutEvents中的。

Amazon Glue 使用适用于 JavaScript (v3) 的 SDK 的示例

以下代码示例向您展示了如何通过使用 Amazon SDK for JavaScript (v3) 来执行操作和实现常见场景 Amazon Glue。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景 是展示如何通过在同一服务中调用多个函数来完成特定任务的代码示例。

每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关如何在上下文中设置和运行代码的说明。

开始使用

你好 Amazon Glue

以下代码示例展示了如何开始使用 Amazon Glue。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { ListJobsCommand, GlueClient } from "@aws-sdk/client-glue";

const client = new GlueClient({});
```

```
export const main = async () => {
  const command = new ListJobsCommand({});

  const { JobNames } = await client.send(command);
  const formattedJobNames = JobNames.join("\n");
  console.log("Job names: ");
  console.log(formattedJobNames);
  return JobNames;
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考ListJobs中的。

主题

- 操作
- 场景

操作

创建爬网程序

以下代码示例显示了如何创建 Amazon Glue 爬虫。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
const createCrawler = (name, role, dbName, tablePrefix, s3TargetPath) => {
  const client = new GlueClient({});

  const command = new CreateCrawlerCommand({
    Name: name,
    Role: role,
    DatabaseName: dbName,
    TablePrefix: tablePrefix,
    Targets: {
```

```
      S3Targets: [{ Path: s3TargetPath }],
    },
  });

  return client.send(command);
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考CreateCrawler中的。

创建任务定义

以下代码示例显示了如何创建 Amazon Glue 作业定义。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
const createJob = (name, role, scriptBucketName, scriptKey) => {
  const client = new GlueClient({});

  const command = new CreateJobCommand({
    Name: name,
    Role: role,
    Command: {
      Name: "glueetl",
      PythonVersion: "3",
      ScriptLocation: `s3://${scriptBucketName}/${scriptKey}`,
    },
    GlueVersion: "3.0",
  });

  return client.send(command);
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考CreateJob中的。

删除爬网程序

以下代码示例显示了如何删除 Amazon Glue 爬虫。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
const deleteCrawler = (crawlerName) => {
  const client = new GlueClient({});

  const command = new DeleteCrawlerCommand({
    Name: crawlerName,
  });

  return client.send(command);
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DeleteCrawler中的。

从 Data Catalog 中删除数据库

以下代码示例展示了如何从 Amazon Glue Data Catalog中删除数据库。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
const deleteDatabase = (databaseName) => {
  const client = new GlueClient({});
```

```
  const command = new DeleteDatabaseCommand({
    Name: databaseName,
  });

  return client.send(command);
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DeleteDatabase中的。

删除任务定义

以下代码示例显示了如何删除 Amazon Glue 作业定义和所有关联的运行。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
const deleteJob = (jobName) => {
  const client = new GlueClient({});

  const command = new DeleteJobCommand({
    JobName: jobName,
  });

  return client.send(command);
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DeleteJob中的。

从数据库中删除表

以下代码示例显示如何从 Amazon Glue Data Catalog 数据库中删除表。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
const deleteTable = (databaseName, tableName) => {
  const client = new GlueClient({});

  const command = new DeleteTableCommand({
    DatabaseName: databaseName,
    Name: tableName,
  });

  return client.send(command);
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DeleteTable中的。

获取爬网程序

以下代码示例显示了如何获取 Amazon Glue 爬虫。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
const getCrawler = (name) => {
  const client = new GlueClient({});

  const command = new GetCrawlerCommand({
    Name: name,
```

```
  });

  return client.send(command);
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考GetCrawler中的。

从 Data Catalog 获取数据库

以下代码示例展示了如何从 Amazon Glue Data Catalog中获取数据库。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
const getDatabase = (name) => {
  const client = new GlueClient({});

  const command = new GetDatabaseCommand({
    Name: name,
  });

  return client.send(command);
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考GetDatabase中的。

运行作业

以下代码示例显示了如何运行作 Amazon Glue 业。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
const getJobRun = (jobName, jobRunId) => {
  const client = new GlueClient({});
  const command = new GetJobRunCommand({
    JobName: jobName,
    RunId: jobRunId,
  });

  return client.send(command);
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考GetJobRun中的。

从 Data Catalog 中获取数据库

以下代码示例展示了如何从 Amazon Glue Data Catalog中获取数据库列表。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
const getDatabases = () => {
  const client = new GlueClient({});

  const command = new GetDatabasesCommand({});

  return client.send(command);
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考GetDatabases中的。

从 Data Catalog 中获取任务

以下代码示例展示了如何从 Amazon Glue Data Catalog中获取作业。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
const getJob = (jobName) => {
  const client = new GlueClient({});

  const command = new GetJobCommand({
    JobName: jobName,
  });

  return client.send(command);
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考GetJob中的。

获取任务运行情况

以下代码示例显示了如何运行作 Amazon Glue 业。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
const getJobRuns = (jobName) => {
  const client = new GlueClient({});
  const command = new GetJobRunsCommand({
    JobName: jobName,
  });

  return client.send(command);
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考GetJobRuns中的。

从数据库获取表

以下代码示例展示了如何在 Amazon Glue Data Catalog中从数据库获取表。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
const getTables = (databaseName) => {
  const client = new GlueClient({});

  const command = new GetTablesCommand({
    DatabaseName: databaseName,
  });

  return client.send(command);
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考GetTables中的。

列出任务定义。

以下代码示例显示了如何列出 Amazon Glue 作业定义。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
const listJobs = () => {
  const client = new GlueClient({});

  const command = new ListJobsCommand({});

  return client.send(command);
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考ListJobs中的。

启动爬网程序

以下代码示例显示了如何启动 Amazon Glue 爬虫。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
const startCrawler = (name) => {
  const client = new GlueClient({});

  const command = new StartCrawlerCommand({
    Name: name,
  });

  return client.send(command);
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考StartCrawler中的。

启动作业运行

以下代码示例显示了如何启动 Amazon Glue 作业运行。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
const startJobRun = (jobName, dbName, tableName, bucketName) => {
  const client = new GlueClient({});

  const command = new StartJobRunCommand({
    JobName: jobName,
    Arguments: {
      "--input_database": dbName,
      "--input_table": tableName,
      "--output_bucket_url": `s3://${bucketName}/`,
    },
  });

  return client.send(command);
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考StartJobRun中的。

场景

爬网程序和作业入门

以下代码示例展示了如何：

- 创建爬网程序，爬取公有 Amazon S3 存储桶并生成包含 CSV 格式的元数据的数据库。

- 列出有关中数据库和表的信息 Amazon Glue Data Catalog。
- 创建任务，从 S3 存储桶提取 CSV 数据，转换数据，然后将 JSON 格式的输出加载到另一个 S3 存储桶中。
- 列出有关作业运行的信息，查看转换后的数据，并清除资源。

有关更多信息，请参阅教程： Amazon Glue Studio 入门。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

创建并运行爬网程序，爬取公共 Amazon Simple Storage Service（Amazon S3）存储桶并生成一个描述其找到的 CSV 格式数据的元数据数据库。

```
const createCrawler = (name, role, dbName, tablePrefix, s3TargetPath) => {
  const client = new GlueClient({});

  const command = new CreateCrawlerCommand({
    Name: name,
    Role: role,
    DatabaseName: dbName,
    TablePrefix: tablePrefix,
    Targets: {
      S3Targets: [{ Path: s3TargetPath }],
    },
  });

  return client.send(command);
};

const getCrawler = (name) => {
  const client = new GlueClient({});

  const command = new GetCrawlerCommand({
    Name: name,
  });
```

```
  return client.send(command);
};

const startCrawler = (name) => {
  const client = new GlueClient({});

  const command = new StartCrawlerCommand({
    Name: name,
  });

  return client.send(command);
};

const crawlerExists = async ({ getCrawler }, crawlerName) => {
  try {
    await getCrawler(crawlerName);
    return true;
  } catch {
    return false;
  }
};

const makeCreateCrawlerStep = (actions) => async (context) => {
  if (await crawlerExists(actions, process.env.CRAWLER_NAME)) {
    log("Crawler already exists. Skipping creation.");
  } else {
    await actions.createCrawler(
      process.env.CRAWLER_NAME,
      process.env.ROLE_NAME,
      process.env.DATABASE_NAME,
      process.env.TABLE_PREFIX,
      process.env.S3_TARGET_PATH
    );

    log("Crawler created successfully.", { type: "success" });
  }

  return { ...context };
};

/**
 * @param {(name: string) => Promise<import('@aws-sdk/client-
glue').GetCrawlerCommandOutput>} getCrawler
 * @param {string} crawlerName
```

```
 */
const waitForCrawler = async (getCrawler, crawlerName) => {
  const waitTimeInSeconds = 30;
  const { Crawler } = await getCrawler(crawlerName);

  if (!Crawler) {
    throw new Error(`Crawler with name ${crawlerName} not found.`);
  }

  if (Crawler.State === "READY") {
    return;
  }

  log(`Crawler is ${Crawler.State}. Waiting ${waitTimeInSeconds} seconds...`);
  await wait(waitTimeInSeconds);
  return waitForCrawler(getCrawler, crawlerName);
};

const makeStartCrawlerStep =
  ({ startCrawler, getCrawler }) =>
  async (context) => {
    log("Starting crawler.");
    await startCrawler(process.env.CRAWLER_NAME);
    log("Crawler started.", { type: "success" });

    log("Waiting for crawler to finish running. This can take a while.");
    await waitForCrawler(getCrawler, process.env.CRAWLER_NAME);
    log("Crawler ready.", { type: "success" });

    return { ...context };
  };
```

列出有关中数据库和表的信息 Amazon Glue Data Catalog。

```
const getDatabase = (name) => {
  const client = new GlueClient({});

  const command = new GetDatabaseCommand({
    Name: name,
  });

  return client.send(command);
```

```
};

const getTables = (databaseName) => {
  const client = new GlueClient({});

  const command = new GetTablesCommand({
    DatabaseName: databaseName,
  });

  return client.send(command);
};

const makeGetDatabaseStep =
  ({ getDatabase }) =>
  async (context) => {
    const {
      Database: { Name },
    } = await getDatabase(process.env.DATABASE_NAME);
    log(`Database: ${Name}`);
    return { ...context };
  };

const makeGetTablesStep =
  ({ getTables }) =>
  async (context) => {
    const { TableList } = await getTables(process.env.DATABASE_NAME);
    log("Tables:");
    log(TableList.map((table) => `  • ${table.Name}\n`));
    return { ...context };
  };
```

创建并运行任务，从源 Amazon S3 存储桶提取 CSV 数据，通过删除和重命名字段对其进行转换，然后将 JSON 格式的输出加载到另一个 Amazon S3 存储桶中。

```
const createJob = (name, role, scriptBucketName, scriptKey) => {
  const client = new GlueClient({});

  const command = new CreateJobCommand({
    Name: name,
    Role: role,
    Command: {
      Name: "glueetl",
```

```
      PythonVersion: "3",
      ScriptLocation: `s3://${scriptBucketName}/${scriptKey}`,
    },
    GlueVersion: "3.0",
  });

  return client.send(command);
};

const startJobRun = (jobName, dbName, tableName, bucketName) => {
  const client = new GlueClient({});

  const command = new StartJobRunCommand({
    JobName: jobName,
    Arguments: {
      "--input_database": dbName,
      "--input_table": tableName,
      "--output_bucket_url": `s3://${bucketName}/`,
    },
  });

  return client.send(command);
};

const makeCreateJobStep =
  ({ createJob }) =>
  async (context) => {
    log("Creating Job.");
    await createJob(
      process.env.JOB_NAME,
      process.env.ROLE_NAME,
      process.env.BUCKET_NAME,
      process.env.PYTHON_SCRIPT_KEY,
    );
    log("Job created.", { type: "success" });

    return { ...context };
  };

/**
 * @param {(name: string, runId: string) => Promise<import('@aws-sdk/client-
glue').GetJobRunCommandOutput> }  getJobRun
 * @param {string} jobName
 * @param {string} jobRunId
```

```
 */
const waitForJobRun = async (getJobRun, jobName, jobRunId) => {
  const waitTimeInSeconds = 30;
  const { JobRun } = await getJobRun(jobName, jobRunId);

  if (!JobRun) {
    throw new Error(`Job run with id ${jobRunId} not found.`);
  }

  switch (JobRun.JobRunState) {
    case "FAILED":
    case "TIMEOUT":
    case "STOPPED":
      throw new Error(
        `Job ${JobRun.JobRunState}. Error: ${JobRun.ErrorMessage}`,
      );
    case "RUNNING":
      break;
    case "SUCCEEDED":
      return;
    default:
      throw new Error(`Unknown job run state: ${JobRun.JobRunState}`);
  }

  log(
    `Job ${JobRun.JobRunState}. Waiting ${waitTimeInSeconds} more seconds...`,
  );
  await wait(waitTimeInSeconds);
  return waitForJobRun(getJobRun, jobName, jobRunId);
};

/**
 * @param {{ prompter: { prompt: () => Promise<{ shouldOpen: boolean }>} }} context
 */
const promptToOpen = async (context) => {
  const { shouldOpen } = await context.prompter.prompt({
    name: "shouldOpen",
    type: "confirm",
    message: "Open the output bucket in your browser?",
  });

  if (shouldOpen) {
    return open(
```

```
      `https://s3.console.aws.amazon.com/s3/buckets/${process.env.BUCKET_NAME} to
 view the output.`,
    );
  }
};

const makeStartJobRunStep =
  ({ startJobRun, getJobRun }) =>
  async (context) => {
    log("Starting job.");
    const { JobRunId } = await startJobRun(
      process.env.JOB_NAME,
      process.env.DATABASE_NAME,
      process.env.TABLE_NAME,
      process.env.BUCKET_NAME,
    );
    log("Job started.", { type: "success" });

    log("Waiting for job to finish running. This can take a while.");
    await waitForJobRun(getJobRun, process.env.JOB_NAME, JobRunId);
    log("Job run succeeded.", { type: "success" });

    await promptToOpen(context);

    return { ...context };
  };
```

列出有关任务运行的信息，并查看一些转换后的数据。

```
const getJobRuns = (jobName) => {
  const client = new GlueClient({});
  const command = new GetJobRunsCommand({
    JobName: jobName,
  });

  return client.send(command);
};

const getJobRun = (jobName, jobRunId) => {
  const client = new GlueClient({});
  const command = new GetJobRunCommand({
    JobName: jobName,
```

```
    RunId: jobRunId,
  });

  return client.send(command);
};

const logJobRunDetails = async (getJobRun, jobName, jobRunId) => {
  const { JobRun } = await getJobRun(jobName, jobRunId);
  log(JobRun, { type: "object" });
};

const makePickJobRunStep =
  ({ getJobRuns, getJobRun }) =>
  async (context) => {
    if (context.selectedJobName) {
      const { JobRuns } = await getJobRuns(context.selectedJobName);

      const { jobRunId } = await context.prompter.prompt({
        name: "jobRunId",
        type: "list",
        message: "Select a job run to see details.",
        choices: JobRuns.map((run) => run.Id),
      });

      logJobRunDetails(getJobRun, context.selectedJobName, jobRunId);
    }

    return { ...context };
  };
```

删除演示创建的所有资源。

```
const deleteJob = (jobName) => {
  const client = new GlueClient({});

  const command = new DeleteJobCommand({
    JobName: jobName,
  });

  return client.send(command);
};
```

```
const deleteTable = (databaseName, tableName) => {
  const client = new GlueClient({});

  const command = new DeleteTableCommand({
    DatabaseName: databaseName,
    Name: tableName,
  });

  return client.send(command);
};

const deleteDatabase = (databaseName) => {
  const client = new GlueClient({});

  const command = new DeleteDatabaseCommand({
    Name: databaseName,
  });

  return client.send(command);
};

const deleteCrawler = (crawlerName) => {
  const client = new GlueClient({});

  const command = new DeleteCrawlerCommand({
    Name: crawlerName,
  });

  return client.send(command);
};

const handleDeleteJobs = async (deleteJobFn, jobNames, context) => {
  const { selectedJobNames } = await context.prompter.prompt({
    name: "selectedJobNames",
    type: "checkbox",
    message: "Let's clean up jobs. Select jobs to delete.",
    choices: jobNames,
  });

  if (selectedJobNames.length === 0) {
    log("No jobs selected.");
  } else {
    log("Deleting jobs.");
    await Promise.all(
```

```
      selectedJobNames.map((n) => deleteJobFn(n).catch(console.error))
    );
    log("Jobs deleted.", { type: "success" });
  }
};

const makeCleanUpJobsStep =
  ({ listJobs, deleteJob }) =>
  async (context) => {
    const { JobNames } = await listJobs();
    if (JobNames.length > 0) {
      await handleDeleteJobs(deleteJob, JobNames, context);
    }

    return { ...context };
  };

const deleteTables = (deleteTable, databaseName, tableNames) =>
  Promise.all(
    tableNames.map((tableName) =>
      deleteTable(databaseName, tableName).catch(console.error)
    )
  );

const makeCleanUpTablesStep =
  ({ getTables, deleteTable }) =>
  async (context) => {
    const { TableList } = await getTables(process.env.DATABASE_NAME).catch(
      () => ({ TableList: null })
    );

    if (TableList && TableList.length > 0) {
      const { tableNames } = await context.prompter.prompt({
        name: "tableNames",
        type: "checkbox",
        message: "Let's clean up tables. Select tables to delete.",
        choices: TableList.map((t) => t.Name),
      });

      if (tableNames.length === 0) {
        log("No tables selected.");
      } else {
        log("Deleting tables.");
        await deleteTables(deleteTable, process.env.DATABASE_NAME, tableNames);
```

```
        log("Tables deleted.", { type: "success" });
      }
    }

    return { ...context };
  };

const deleteDatabases = (deleteDatabase, databaseNames) =>
  Promise.all(
    databaseNames.map((dbName) => deleteDatabase(dbName).catch(console.error))
  );

const makeCleanUpDatabasesStep =
  ({ getDatabases, deleteDatabase }) =>
  async (context) => {
    const { DatabaseList } = await getDatabases();

    if (DatabaseList.length > 0) {
      const { dbNames } = await context.prompter.prompt({
        name: "dbNames",
        type: "checkbox",
        message: "Let's clean up databases. Select databases to delete.",
        choices: DatabaseList.map((db) => db.Name),
      });

      if (dbNames.length === 0) {
        log("No databases selected.");
      } else {
        log("Deleting databases.");
        await deleteDatabases(deleteDatabase, dbNames);
        log("Databases deleted.", { type: "success" });
      }
    }

    return { ...context };
  };

const cleanUpCrawlerStep = async (context) => {
  log(`Deleting crawler.`);

  try {
    await deleteCrawler(process.env.CRAWLER_NAME);
    log("Crawler deleted.", { type: "success" });
  } catch (err) {
```

```
    if (err.name === "EntityNotFoundException") {
      log(`Crawler is already deleted.`);
    } else {
      throw err;
    }
  }

  return { ...context };
};
```

- 有关 API 详细信息，请参阅《Amazon SDK for JavaScript API 参考》中的以下主题。
  - CreateCrawler
  - CreateJob
  - DeleteCrawler
  - DeleteDatabase
  - DeleteJob
  - DeleteTable
  - GetCrawler
  - GetDatabase
  - GetDatabases
  - GetJob
  - GetJobRun
  - GetJobRuns
  - GetTables
  - ListJobs
  - StartCrawler
  - StartJobRun

HealthImaging 使用适用于 JavaScript (v3) 的 SDK 的示例

以下代码示例向您展示了如何通过使用 Amazon SDK for JavaScript (v3) 来执行操作和实现常见场景 HealthImaging。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景 是展示如何通过在同一服务中调用多个函数来完成特定任务的代码示例。

每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关如何在上下文中设置和运行代码的说明。

开始使用

你好 HealthImaging

以下代码示例展示了如何开始使用 HealthImaging。

适用于 JavaScript (v3) 的软件开发工具包

```
import {
  ListDatastoresCommand,
  MedicalImagingClient,
} from "@aws-sdk/client-medical-imaging";

// When no region or credentials are provided, the SDK will use the
// region and credentials from the local AWS config.
const client = new MedicalImagingClient({});

export const helloMedicalImaging = async () => {
  const command = new ListDatastoresCommand({});

  const { datastoreSummaries } = await client.send(command);
  console.log("Datastores: ");
  console.log(datastoreSummaries.map((item) => item.datastoreName).join("\n"));
  return datastoreSummaries;
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考ListDatastores中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

主题

- 操作
- 场景

操作

将标签添加到资源中

以下代码示例显示了如何为 HealthImaging 资源添加标签。

适用于 JavaScript (v3) 的软件开发工具包

```
import { TagResourceCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data store
 or image set.
 * @param {Record<string,string>} tags - The tags to add to the resource as JSON.
 *                      - For example: {"Deployment" : "Development"}
 */
export const tagResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:xxxxxx:datastore/xxxxx/imageset/
xxx",
  tags = {}
) => {
  const response = await medicalImagingClient.send(
    new TagResourceCommand({ resourceArn: resourceArn, tags: tags })
  );
  console.log(response);
  // {
  //     '$metadata': {
  //        httpStatusCode: 204,
  //         requestId: '8a6de9a3-ec8e-47ef-8643-473518b19d45',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //     }
  // }

  return response;
```

```
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考TagResource中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

复制映像集

以下代码示例显示了如何复制 HealthImaging 图像集。

适用于 JavaScript (v3) 的软件开发工具包

用于复制映像集的实用程序函数。

```
import { CopyImageSetCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} imageSetId - The source image set ID.
 * @param {string} sourceVersionId - The source version ID.
 * @param {string} destinationImageSetId - The optional ID of the destination image
 set.
 * @param {string} destinationVersionId - The optional version ID of the destination
 image set.
 */
export const copyImageSet = async (
  datastoreId = "xxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxxx",
  sourceVersionId = "1",
  destinationImageSetId = "",
  destinationVersionId = ""
) => {
  const params = {
    datastoreId: datastoreId,
    sourceImageSetId: imageSetId,
    copyImageSetInformation: {
      sourceImageSet: { latestVersionId: sourceVersionId },
```

```
    },
  };
  if (destinationImageSetId !== "" && destinationVersionId !== "") {
    params.copyImageSetInformation.destinationImageSet = {
      imageSetId: destinationImageSetId,
      latestVersionId: destinationVersionId,
    };
  }

  const response = await medicalImagingClient.send(
    new CopyImageSetCommand(params)
  );
  console.log(response);
  // {
  //     '$metadata': {
  //         httpStatusCode: 200,
  //         requestId: 'd9b219ce-cc48-4a44-a5b2-c5c3068f1ee8',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //      },
  //       datastoreId: 'xxxxxxxxxxxxxx',
  //       destinationImageSetProperties: {
  //             createdAt: 2023-09-27T19:46:21.824Z,
  //             imageSetArn: 'arn:aws:medical-imaging:us-
east-1:xxxxxxxxxxx:datastore/xxxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxxxxxx',
  //             imageSetId: 'xxxxxxxxxxxxxxx',
  //             imageSetState: 'LOCKED',
  //             imageSetWorkflowStatus: 'COPYING',
  //             latestVersionId: '1',
  //             updatedAt: 2023-09-27T19:46:21.824Z
  //       },
  //       sourceImageSetProperties: {
  //             createdAt: 2023-09-22T14:49:26.427Z,
  //             imageSetArn: 'arn:aws:medical-imaging:us-
east-1:xxxxxxxxxxx:datastore/xxxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxxx',
  //             imageSetId: 'xxxxxxxxxxxxxxxx',
  //             imageSetState: 'LOCKED',
  //             imageSetWorkflowStatus: 'COPYING_WITH_READ_ONLY_ACCESS',
  //             latestVersionId: '4',
  //             updatedAt: 2023-09-27T19:46:21.824Z
  //      }
  // }
```

```
  return response;
};
```

复制没有目标的映像集。

```
  try {
    await copyImageSet(
      "12345678901234567890123456789012",
      "12345678901234567890123456789012",
      "1"
    );
  } catch (err) {
    console.error(err);
  }
```

复制带有目标的映像集。

```
  try {
    await copyImageSet(
      "12345678901234567890123456789012",
      "12345678901234567890123456789012",
      "4",
      "12345678901234567890123456789012",
      "1"
    );
  } catch (err) {
    console.error(err);
  }
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考CopyImageSet中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

创建数据存储

以下代码示例显示了如何创建 HealthImaging 数据存储。

适用于 JavaScript (v3) 的软件开发工具包

```
import { CreateDatastoreCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreName - The name of the data store to create.
 */
export const createDatastore = async (datastoreName = "DATASTORE_NAME") => {
  const response = await medicalImagingClient.send(
    new CreateDatastoreCommand({ datastoreName: datastoreName })
  );
  console.log(response);
  // {
  //   '$metadata': {
  //       httpStatusCode: 200,
  //       requestId: 'a71cd65f-2382-49bf-b682-f9209d8d399b',
  //       extendedRequestId: undefined,
  //       cfId: undefined,
  //       attempts: 1,
  //       totalRetryDelay: 0
  //    },
  //    datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //    datastoreStatus: 'CREATING'
  // }
  return response;
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考CreateDatastore中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

删除数据存储

以下代码示例显示了如何删除 HealthImaging 数据存储。

适用于 JavaScript (v3) 的软件开发工具包

```
import { DeleteDatastoreCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the data store to delete.
 */
export const deleteDatastore = async (datastoreId = "DATASTORE_ID") => {
  const response = await medicalImagingClient.send(
    new DeleteDatastoreCommand({ datastoreId })
  );
  console.log(response);
  // {
  //   '$metadata': {
  //           httpStatusCode: 200,
  //           requestId: 'f5beb409-678d-48c9-9173-9a001ee1ebb1',
  //           extendedRequestId: undefined,
  //           cfId: undefined,
  //           attempts: 1,
  //           totalRetryDelay: 0
  //        },
  //     datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //     datastoreStatus: 'DELETING'
  // }

  return response;
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DeleteDatastore中的。

> Note
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

删除映像集

以下代码示例显示了如何删除 HealthImaging 影像集。

适用于 JavaScript (v3) 的软件开发工具包

```
import { DeleteImageSetCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The data store ID.
 * @param {string} imageSetId - The image set ID.
 */
export const deleteImageSet = async (
  datastoreId = "xxxxxxxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxxxxxxx"
) => {
  const response = await medicalImagingClient.send(
    new DeleteImageSetCommand({
      datastoreId: datastoreId,
      imageSetId: imageSetId,
    })
  );
  console.log(response);
  // {
  //     '$metadata': {
  //          httpStatusCode: 200,
  //          requestId: '6267bbd2-eaa5-4a50-8ee8-8fddf535cf73',
  //          extendedRequestId: undefined,
  //          cfId: undefined,
  //          attempts: 1,
  //          totalRetryDelay: 0
  //     },
  //     datastoreId: 'xxxxxxxxxxxxxxxx',
  //     imageSetId: 'xxxxxxxxxxxxxxx',
  //     imageSetState: 'LOCKED',
  //     imageSetWorkflowStatus: 'DELETING'
  // }
  return response;
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DeleteImageSet中的。

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

获取映像帧

以下代码示例展示了如何获取映像帧。

适用于 JavaScript (v3) 的软件开发工具包

```
import { GetImageFrameCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} imageFrameFileName - The name of the file for the HTJ2K-encoded
 image frame.
 * @param {string} datastoreID - The data store's ID.
 * @param {string} imageSetID - The image set's ID.
 * @param {string} imageFrameID - The image frame's ID.
 */
export const getImageFrame = async (
  imageFrameFileName = "image.jph",
  datastoreID = "DATASTORE_ID",
  imageSetID = "IMAGE_SET_ID",
  imageFrameID = "IMAGE_FRAME_ID"
) => {
  const response = await medicalImagingClient.send(
    new GetImageFrameCommand({
      datastoreId: datastoreID,
      imageSetId: imageSetID,
      imageFrameInformation: { imageFrameId: imageFrameID },
    })
  );
  const buffer = await response.imageFrameBlob.transformToByteArray();
  writeFileSync(imageFrameFileName, buffer);

  console.log(response);
  // {
  //     '$metadata': {
  //         httpStatusCode: 200,
  //         requestId: 'e4ab42a5-25a3-4377-873f-374ecf4380e1',
```

```
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //     },
  //     contentType: 'application/octet-stream',
  //     imageFrameBlob: <ref *1> IncomingMessage {}
  // }
  return response;
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考GetImageFrame中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

获取数据存储属性

以下代码示例显示了如何获取 HealthImaging 数据存储属性。

适用于 JavaScript (v3) 的软件开发工具包

```
import { GetDatastoreCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreID - The ID of the data store.
 */
export const getDatastore = async (datastoreID = "DATASTORE_ID") => {
  const response = await medicalImagingClient.send(
    new GetDatastoreCommand({ datastoreId: datastoreID })
  );
  console.log(response);
  // {
  //   '$metadata': {
  //       httpStatusCode: 200,
  //       requestId: '55ea7d2e-222c-4a6a-871e-4f591f40cadb',
  //       extendedRequestId: undefined,
  //       cfId: undefined,
```

```
  //       attempts: 1,
  //       totalRetryDelay: 0
  //    },
  //   datastoreProperties: {
  //        createdAt: 2023-08-04T18:50:36.239Z,
  //         datastoreArn: 'arn:aws:medical-imaging:us-east-1:xxxxxxxxx:datastore/
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //         datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //         datastoreName: 'my_datastore',
  //         datastoreStatus: 'ACTIVE',
  //         updatedAt: 2023-08-04T18:50:36.239Z
  //   }
  // }
  return response["datastoreProperties"];
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考GetDatastore中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

获取映像集属性

以下代码示例说明如何获取 HealthImaging 影像集属性。

适用于 JavaScript (v3) 的软件开发工具包

```
import { GetImageSetCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} imageSetId - The ID of the image set.
 * @param {string} imageSetVersion - The optional version of the image set.
 *
 */
export const getImageSet = async (
  datastoreId = "xxxxxxxxxxxxxxx",
```

```
  imageSetId = "xxxxxxxxxxxxxxx",
  imageSetVersion = ""
) => {
  let params = { datastoreId: datastoreId, imageSetId: imageSetId };
  if (imageSetVersion !== "") {
    params.imageSetVersion = imageSetVersion;
  }
  const response = await medicalImagingClient.send(
    new GetImageSetCommand(params)
  );
  console.log(response);
  // {
  //     '$metadata': {
  //     httpStatusCode: 200,
  //         requestId: '0615c161-410d-4d06-9d8c-6e1241bb0a5a',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  // },
  //     createdAt: 2023-09-22T14:49:26.427Z,
  //     datastoreId: 'xxxxxxxxxxxxxxx',
  //     imageSetArn: 'arn:aws:medical-imaging:us-east-1:xxxxxxxxxx:datastore/
xxxxxxxxxxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxxxxxxx',
  //     imageSetId: 'xxxxxxxxxxxxxxx',
  //     imageSetState: 'ACTIVE',
  //     imageSetWorkflowStatus: 'CREATED',
  //     updatedAt: 2023-09-22T14:49:26.427Z,
  //     versionId: '1'
  // }

  return response;
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考GetImageSet中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

获取导入任务属性

以下代码示例展示了如何获取导入作业属性。

适用于 JavaScript (v3) 的软件开发工具包

```
import { GetDICOMImportJobCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} jobId - The ID of the import job.
 */
export const getDICOMImportJob = async (
  datastoreId = "xxxxxxxxxxxxxxxxxxxx",
  jobId = "xxxxxxxxxxxxxxxxxxxx"
) => {
  const response = await medicalImagingClient.send(
    new GetDICOMImportJobCommand({ datastoreId: datastoreId, jobId: jobId })
  );
  console.log(response);
  // {
  //     '$metadata': {
  //     httpStatusCode: 200,
  //         requestId: 'a2637936-78ea-44e7-98b8-7a87d95dfaee',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  // },
  //     jobProperties: {
  //         dataAccessRoleArn: 'arn:aws:iam::xxxxxxxxxxxx:role/dicom_import',
  //             datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxx',
  //             endedAt: 2023-09-19T17:29:21.753Z,
  //             inputS3Uri: 's3://healthimaging-source/CTStudy/',
  //             jobId: ''xxxxxxxxxxxxxxxxxxxxxxxxx'',
  //             jobName: 'job_1',
  //             jobStatus: 'COMPLETED',
  //             outputS3Uri: 's3://health-imaging-dest/
ouput_ct/'xxxxxxxxxxxxxxxxxxxxxxxxx'-DicomImport-'xxxxxxxxxxxxxxxxxxxxxxxxx'/',
  //             submittedAt: 2023-09-19T17:27:25.143Z
  //     }
  // }
```

```
  return response;
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript AP I 参考ImportJob中的 getDicom。

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

获取映像集的元数据

以下代码示例说明如何获取 HealthImaging 影像集的元数据。

适用于 JavaScript (v3) 的软件开发工具包

用于获取映像集元数据的实用程序函数。

```
import { GetImageSetMetadataCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";
import { writeFileSync } from "fs";

/**
 * @param {string} metadataFileName - The name of the file for the gzipped metadata.
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} imagesetId - The ID of the image set.
 * @param {string} versionID - The optional version ID of the image set.
 */
export const getImageSetMetadata = async (
  metadataFileName = "metadata.json.gzip",
  datastoreId = "xxxxxxxxxxxxxx",
  imagesetId = "xxxxxxxxxxxxxx",
  versionID = ""
) => {
  const params = { datastoreId: datastoreId, imageSetId: imagesetId };

  if (versionID) {
    params.versionID = versionID;
  }

  const response = await medicalImagingClient.send(
```

```
    new GetImageSetMetadataCommand(params)
  );
  const buffer = await response.imageSetMetadataBlob.transformToByteArray();
  writeFileSync(metadataFileName, buffer);

  console.log(response);
  // {
  //     '$metadata': {
  //     httpStatusCode: 200,
  //         requestId: '5219b274-30ff-4986-8cab-48753de3a599',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  // },
  //     contentType: 'application/json',
  //     contentEncoding: 'gzip',
  //     imageSetMetadataBlob: <ref *1> IncomingMessage {}
  // }

  return response;
};
```

获取没有版本的映像集元数据。

```
  try {
    await getImageSetMetadata(
      "metadata.json.gzip",
      "12345678901234567890123456789012",
      "12345678901234567890123456789012"
    );
  } catch (err) {
    console.log("Error", err);
  }
```

获取带有版本的映像集元数据。

```
  try {
    await getImageSetMetadata(
      "metadata2.json.gzip",
```

```
      "12345678901234567890123456789012",
      "12345678901234567890123456789012",
      "1"
    );
  } catch (err) {
    console.log("Error", err);
  }
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考GetImageSetMetadata中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

将批量数据导入数据存储

以下代码示例说明如何将批量数据导入 HealthImaging 数据存储。

适用于 JavaScript (v3) 的软件开发工具包

```
import { StartDICOMImportJobCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} jobName - The name of the import job.
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} dataAccessRoleArn - The Amazon Resource Name (ARN) of the role
 that grants permission.
 * @param {string} inputS3Uri - The URI of the S3 bucket containing the input files.
 * @param {string} outputS3Uri - The URI of the S3 bucket where the output files are
 stored.
 */
export const startDicomImportJob = async (
  jobName = "test-1",
  datastoreId = "12345678901234567890123456789012",
  dataAccessRoleArn = "arn:aws:iam::xxxxxxxxxxxx:role/ImportJobDataAccessRole",
  inputS3Uri = "s3://medical-imaging-dicom-input/dicom_input/",
  outputS3Uri = "s3://medical-imaging-output/job_output/"
```

```
) => {
  const response = await medicalImagingClient.send(
    new StartDICOMImportJobCommand({
      jobName: jobName,
      datastoreId: datastoreId,
      dataAccessRoleArn: dataAccessRoleArn,
      inputS3Uri: inputS3Uri,
      outputS3Uri: outputS3Uri,
    })
  );
  console.log(response);
  // {
  //     '$metadata': {
  //     httpStatusCode: 200,
  //         requestId: '6e81d191-d46b-4e48-a08a-cdcc7e11eb79',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  // },
  //     datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //     jobId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //     jobStatus: 'SUBMITTED',
  //     submittedAt: 2023-09-22T14:48:45.767Z
  // }
  return response;
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript AP I 参考ImportJob中的 StartDicom。

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

列出数据存储

以下代码示例显示了如何列出 HealthImaging 数据存储。

适用于 JavaScript (v3) 的软件开发工具包

```
import { paginateListDatastores } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

export const listDatastores = async () => {
  const paginatorConfig = {
    client: medicalImagingClient,
    pageSize: 50,
  };

  const commandParams = {};
  const paginator = paginateListDatastores(paginatorConfig, commandParams);

  /**
   * @type {import("@aws-sdk/client-medical-imaging").DatastoreSummary[]}
   */
  const datastoreSummaries = [];
  for await (const page of paginator) {
    // Each page contains a list of `jobSummaries`. The list is truncated if is
 larger than `pageSize`.
    datastoreSummaries.push(...page["datastoreSummaries"]);
    console.log(page);
  }
  // {
  //   '$metadata': {
  //       httpStatusCode: 200,
  //       requestId: '6aa99231-d9c2-4716-a46e-edb830116fa3',
  //       extendedRequestId: undefined,
  //       cfId: undefined,
  //       attempts: 1,
  //       totalRetryDelay: 0
  //   },
  //   datastoreSummaries: [
  //     {
  //       createdAt: 2023-08-04T18:49:54.429Z,
  //       datastoreArn: 'arn:aws:medical-imaging:us-east-1:xxxxxxxxx:datastore/
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //       datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //       datastoreName: 'my_datastore',
  //       datastoreStatus: 'ACTIVE',
  //       updatedAt: 2023-08-04T18:49:54.429Z
  //     }
  //     ...
```

```
  //   ]
  // }

  return datastoreSummaries;
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考ListDatastores中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

列出图像集版本

以下代码示例显示了如何列出 HealthImaging 影像集版本。

适用于 JavaScript (v3) 的软件开发工具包

```
import { paginateListImageSetVersions } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} imageSetId - The ID of the image set.
 */
export const listImageSetVersions = async (
  datastoreId = "xxxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxxx"
) => {
  const paginatorConfig = {
    client: medicalImagingClient,
    pageSize: 50,
  };

  const commandParams = { datastoreId, imageSetId };
  const paginator = paginateListImageSetVersions(
    paginatorConfig,
    commandParams
  );
```

```
  let imageSetPropertiesList = [];
  for await (const page of paginator) {
    // Each page contains a list of `jobSummaries`. The list is truncated if is
 larger than `pageSize`.
    imageSetPropertiesList.push(...page["imageSetPropertiesList"]);
    console.log(page);
  }
  // {
  //     '$metadata': {
  //         httpStatusCode: 200,
  //         requestId: '74590b37-a002-4827-83f2-3c590279c742',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //     },
  //     imageSetPropertiesList: [
  //         {
  //             ImageSetWorkflowStatus: 'CREATED',
  //             createdAt: 2023-09-22T14:49:26.427Z,
  //             imageSetId: 'xxxxxxxxxxxxxxxxxxxxxxx',
  //             imageSetState: 'ACTIVE',
  //             versionId: '1'
  //         }]
  // }
  return imageSetPropertiesList;
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考ListImageSetVersions中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

列出数据存储的导入任务

以下代码示例显示了如何列出 HealthImaging 数据存储的导入任务。

适用于 JavaScript (v3) 的软件开发工具包

```
import { paginateListDICOMImportJobs } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the data store.
 */
export const listDICOMImportJobs = async (
  datastoreId = "xxxxxxxxxxxxxxxxxx"
) => {
  const paginatorConfig = {
    client: medicalImagingClient,
    pageSize: 50,
  };

  const commandParams = { datastoreId: datastoreId };
  const paginator = paginateListDICOMImportJobs(paginatorConfig, commandParams);

  let jobSummaries = [];
  for await (const page of paginator) {
    // Each page contains a list of `jobSummaries`. The list is truncated if is
 larger than `pageSize`.
    jobSummaries.push(...page["jobSummaries"]);
    console.log(page);
  }
  // {
  //     '$metadata': {
  //     httpStatusCode: 200,
  //         requestId: '3c20c66e-0797-446a-a1d8-91b742fd15a0',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  // },
  //     jobSummaries: [
  //         {
  //             dataAccessRoleArn: 'arn:aws:iam::xxxxxxxxxxxx:role/dicom_import',
  //             datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxx',
  //             endedAt: 2023-09-22T14:49:51.351Z,
  //             jobId: 'xxxxxxxxxxxxxxxxxxxxxxxxx',
  //             jobName: 'test-1',
  //             jobStatus: 'COMPLETED',
  //             submittedAt: 2023-09-22T14:48:45.767Z
```

```
  // }
  // ]}

  return jobSummaries;
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript AP I 参考ImportJobs中的 ListDicom。

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

列出资源的标签

以下代码示例显示了如何列出 HealthImaging 资源的标签。

适用于 JavaScript (v3) 的软件开发工具包

```
import { ListTagsForResourceCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data store
 or image set.
 */
export const listTagsForResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:abc:datastore/def/imageset/ghi"
) => {
  const response = await medicalImagingClient.send(
    new ListTagsForResourceCommand({ resourceArn: resourceArn })
  );
  console.log(response);
  // {
  //     '$metadata': {
  //         httpStatusCode: 200,
  //         requestId: '008fc6d3-abec-4870-a155-20fa3631e645',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
```

```
  //         attempts: 1,
  //         totalRetryDelay: 0
  //     },
  //     tags: { Deployment: 'Development' }
  // }

  return response;
};
```

• 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考ListTagsForResource中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

从资源中删除标签

以下代码示例显示了如何从 HealthImaging 资源中移除标签。

适用于 JavaScript (v3) 的软件开发工具包

```
import { UntagResourceCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data store
 or image set.
 * @param {string[]} tagKeys - The keys of the tags to remove.
 */
export const untagResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:xxxxxx:datastore/xxxxx/imageset/
xxx",
  tagKeys = []
) => {
  const response = await medicalImagingClient.send(
    new UntagResourceCommand({ resourceArn: resourceArn, tagKeys: tagKeys })
  );
  console.log(response);
```

```
  // {
  //     '$metadata': {
  //        httpStatusCode: 204,
  //         requestId: '8a6de9a3-ec8e-47ef-8643-473518b19d45',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //    }
  // }

  return response;
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考UntagResource中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

搜索映像集

以下代码示例显示了如何搜索 HealthImaging 影像集。

适用于 JavaScript (v3) 的软件开发工具包

用于搜索映像集的实用程序函数。

```
import { paginateSearchImageSets } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The data store's ID.
 * @param { import('@aws-sdk/client-medical-imaging').SearchFilter[] } filters - The
 search criteria filters.
 */
export const searchImageSets = async (
  datastoreId = "xxxxxxxx",
  filters = []
) => {
```

```
  const paginatorConfig = {
    client: medicalImagingClient,
    pageSize: 50,
  };

  const commandParams = {
    datastoreId: datastoreId,
    searchCriteria: {
      filters,
    },
  };

  const paginator = paginateSearchImageSets(paginatorConfig, commandParams);

  const imageSetsMetadataSummaries = [];
  for await (const page of paginator) {
    // Each page contains a list of `jobSummaries`. The list is truncated if is
 larger than `pageSize`.
    imageSetsMetadataSummaries.push(...page["imageSetsMetadataSummaries"]);
    console.log(page);
  }
  // {
  //     '$metadata': {
  //         httpStatusCode: 200,
  //         requestId: 'f009ea9c-84ca-4749-b5b6-7164f00a5ada',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //     },
  //     imageSetsMetadataSummaries: [
  //         {
  //             DICOMTags: [Object],
  //             createdAt: "2023-09-19T16:59:40.551Z",
  //             imageSetId: '7f75e1b5c0f40eac2b24cf712f485f50',
  //             updatedAt: "2023-09-19T16:59:40.551Z",
  //             version: 1
  //         }]
  // }

  return imageSetsMetadataSummaries;
};
```

使用案例 #1：EQUAL 运算符。

```
const datastoreId = "12345678901234567890123456789012";

try {
  const filters = [
    {
      values: [{ DICOMPatientId: "9227465" }],
      operator: "EQUAL",
    },
  ];

  await searchImageSets(datastoreId, filters);
} catch (err) {
  console.error(err);
}
```

用例 #2: 在使用 DICOM 和 DICOM 的运算符之间StudyDate 。StudyTime

```
const datastoreId = "12345678901234567890123456789012";

try {
  const filters = [
    {
      values: [
        {
          DICOMStudyDateAndTime: {
            DICOMStudyDate: "19900101",
            DICOMStudyTime: "000000",
          },
        },
        {
          DICOMStudyDateAndTime: {
            DICOMStudyDate: "20230901",
            DICOMStudyTime: "000000",
          },
        },
      ],
      operator: "BETWEEN",
    },
  ];
```

```
    await searchImageSets(datastoreId, filters);
  } catch (err) {
    console.error(err);
  }
```

使用案例 #3：使用 createdAt 的 BETWEEN 运算符。时间研究以前一直存在。

```
  const datastoreId = "12345678901234567890123456789012";

  try {
    const filters = [
      {
        values: [
          { createdAt: new Date("1985-04-12T23:20:50.52Z") },
          { createdAt: new Date("2023-09-12T23:20:50.52Z") },
        ],
        operator: "BETWEEN",
      },
    ];

    await searchImageSets(datastoreId, filters);
  } catch (err) {
    console.error(err);
  }
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考SearchImageSets中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

更新映像集元数据

以下代码示例显示如何更新 HealthImaging 影像集元数据。

适用于 JavaScript (v3) 的软件开发工具包

```
import {UpdateImageSetMetadataCommand} from "@aws-sdk/client-medical-imaging";
```

```
import {medicalImagingClient} from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the HealthImaging data store.
 * @param {string} imageSetId - The ID of the HealthImaging image set.
 * @param {string} latestVersionId - The ID of the HealthImaging image set version.
 * @param {{}} updateMetadata - The metadata to update.
 */
export const updateImageSetMetadata = async (datastoreId = "xxxxxxxxxx",
                                             imageSetId = "xxxxxxxxxx",
                                             latestVersionId = "1",
                                             updateMetadata = '{}') => {
    const response = await medicalImagingClient.send(
        new UpdateImageSetMetadataCommand({
            datastoreId: datastoreId,
            imageSetId: imageSetId,
            latestVersionId: latestVersionId,
            updateImageSetMetadataUpdates: updateMetadata
        })
    );
    console.log(response);
    // {
    //     '$metadata': {
    //     httpStatusCode: 200,
    //         requestId: '7966e869-e311-4bff-92ec-56a61d3003ea',
    //         extendedRequestId: undefined,
    //         cfId: undefined,
    //         attempts: 1,
    //         totalRetryDelay: 0
    // },
    //     createdAt: 2023-09-22T14:49:26.427Z,
    //     datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
    //     imageSetId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
    //     imageSetState: 'LOCKED',
    //     imageSetWorkflowStatus: 'UPDATING',
    //     latestVersionId: '4',
    //     updatedAt: 2023-09-27T19:41:43.494Z
    // }
    return response;
};
```

对元数据进行编码。

```
    const updatableAttributes =
JSON.stringify({
  "SchemaVersion": 1.1,
  "Patient": {
    "DICOM": {
      "PatientName": "Garcia^Gloria"
    }
  }
})

    const updateMetadata = {
        "DICOMUpdates": {
            "updatableAttributes":
                new TextEncoder().encode(updatableAttributes)
        }
    };

    await updateImageSetMetadata("12345678901234567890123456789012",
 "12345678901234567890123456789012",
        "1", updateMetadata);
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考UpdateImageSetMetadata中的。

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

场景

标记数据存储

以下代码示例显示了如何为 HealthImaging 数据存储添加标签。

适用于 JavaScript (v3) 的软件开发工具包

标记数据存储。

```
  try {
```

```
    const datastoreArn =
      "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012";
    const tags = {
      Deployment: "Development",
    };
    await tagResource(datastoreArn, tags);
  } catch (e) {
    console.log(e);
  }
```

用于标记资源的实用程序函数。

```
import { TagResourceCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data store
 or image set.
 * @param {Record<string,string>} tags - The tags to add to the resource as JSON.
 *                      - For example: {"Deployment" : "Development"}
 */
export const tagResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:xxxxxx:datastore/xxxxx/imageset/
xxx",
  tags = {}
) => {
  const response = await medicalImagingClient.send(
    new TagResourceCommand({ resourceArn: resourceArn, tags: tags })
  );
  console.log(response);
  // {
  //     '$metadata': {
  //        httpStatusCode: 204,
  //         requestId: '8a6de9a3-ec8e-47ef-8643-473518b19d45',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //    }
  // }
```

```
  return response;
};
```

列出数据存储的标签。

```
  try {
    const datastoreArn =
      "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012";
    const { tags } = await listTagsForResource(datastoreArn);
    console.log(tags);
  } catch (e) {
    console.log(e);
  }
```

用于列出资源标签的实用程序函数。

```
import { ListTagsForResourceCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data store
 or image set.
 */
export const listTagsForResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:abc:datastore/def/imageset/ghi"
) => {
  const response = await medicalImagingClient.send(
    new ListTagsForResourceCommand({ resourceArn: resourceArn })
  );
  console.log(response);
  // {
  //     '$metadata': {
  //         httpStatusCode: 200,
  //         requestId: '008fc6d3-abec-4870-a155-20fa3631e645',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //     },
  //     tags: { Deployment: 'Development' }
```

```
  // }

  return response;
};
```

取消标记数据存储。

```
  try {
    const datastoreArn =
      "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012";
    const keys = ["Deployment"];
    await untagResource(datastoreArn, keys);
  } catch (e) {
    console.log(e);
  }
```

用于取消标记资源的实用程序函数。

```
import { UntagResourceCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data store
 or image set.
 * @param {string[]} tagKeys - The keys of the tags to remove.
 */
export const untagResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:xxxxxx:datastore/xxxxx/imageset/
xxx",
  tagKeys = []
) => {
  const response = await medicalImagingClient.send(
    new UntagResourceCommand({ resourceArn: resourceArn, tagKeys: tagKeys })
  );
  console.log(response);
  // {
  //     '$metadata': {
  //        httpStatusCode: 204,
  //         requestId: '8a6de9a3-ec8e-47ef-8643-473518b19d45',
  //         extendedRequestId: undefined,
```

```
  //          cfId: undefined,
  //          attempts: 1,
  //          totalRetryDelay: 0
  //     }
  // }

  return response;
};
```

- 有关 API 详细信息，请参阅《Amazon SDK for JavaScript API 参考》中的以下主题。
  - ListTagsForResource
  - TagResource
  - UntagResource

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

标记映像集

以下代码示例显示了如何为 HealthImaging 图像集添加标签。

适用于 JavaScript (v3) 的软件开发工具包

标记映像集。

```
  try {
    const imagesetArn =
      "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012/
imageset/12345678901234567890123456789012";
    const tags = {
      Deployment: "Development",
    };
    await tagResource(imagesetArn, tags);
  } catch (e) {
    console.log(e);
  }
```

用于标记资源的实用程序函数。

```
import { TagResourceCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data store
 or image set.
 * @param {Record<string,string>} tags - The tags to add to the resource as JSON.
 *                     - For example: {"Deployment" : "Development"}
 */
export const tagResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:xxxxxx:datastore/xxxxx/imageset/
xxx",
  tags = {}
) => {
  const response = await medicalImagingClient.send(
    new TagResourceCommand({ resourceArn: resourceArn, tags: tags })
  );
  console.log(response);
  // {
  //     '$metadata': {
  //        httpStatusCode: 204,
  //         requestId: '8a6de9a3-ec8e-47ef-8643-473518b19d45',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //    }
  // }

  return response;
};
```

列出映像集的标签。

```
  try {
    const imagesetArn =
```

```
      "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012/
imageset/12345678901234567890123456789012";
    const { tags } = await listTagsForResource(imagesetArn);
    console.log(tags);
  } catch (e) {
    console.log(e);
  }
```

用于列出资源标签的实用程序函数。

```
import { ListTagsForResourceCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data store
 or image set.
 */
export const listTagsForResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:abc:datastore/def/imageset/ghi"
) => {
  const response = await medicalImagingClient.send(
    new ListTagsForResourceCommand({ resourceArn: resourceArn })
  );
  console.log(response);
  // {
  //     '$metadata': {
  //         httpStatusCode: 200,
  //         requestId: '008fc6d3-abec-4870-a155-20fa3631e645',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //     },
  //     tags: { Deployment: 'Development' }
  // }

  return response;
};
```

取消标记映像集。

```
  try {
    const imagesetArn =
      "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012/
imageset/12345678901234567890123456789012";
    const keys = ["Deployment"];
    await untagResource(imagesetArn, keys);
  } catch (e) {
    console.log(e);
  }
```

用于取消标记资源的实用程序函数。

```
import { UntagResourceCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data store
 or image set.
 * @param {string[]} tagKeys - The keys of the tags to remove.
 */
export const untagResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:xxxxxx:datastore/xxxxx/imageset/
xxx",
  tagKeys = []
) => {
  const response = await medicalImagingClient.send(
    new UntagResourceCommand({ resourceArn: resourceArn, tagKeys: tagKeys })
  );
  console.log(response);
  // {
  //     '$metadata': {
  //        httpStatusCode: 204,
  //         requestId: '8a6de9a3-ec8e-47ef-8643-473518b19d45',
  //         extendedRequestId: undefined,
  //         cfId: undefined,
  //         attempts: 1,
  //         totalRetryDelay: 0
  //    }
  // }

  return response;
```

```
};
```

- 有关 API 详细信息，请参阅《Amazon SDK for JavaScript API 参考》中的以下主题。
  - ListTagsForResource
  - TagResource
  - UntagResource

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

使用适用于 JavaScript (v3) 的开发工具包的 IAM 示例

以下代码示例向您展示了如何使用带有 IAM 的 Amazon SDK for JavaScript (v3) 来执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景 是展示如何通过在同一服务中调用多个函数来完成特定任务的代码示例。

每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关如何在上下文中设置和运行代码的说明。

开始使用

开始使用 IAM

以下代码示例展示了如何开始使用 IAM。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { IAMClient, paginateListPolicies } from "@aws-sdk/client-iam";

const client = new IAMClient({});

export const listLocalPolicies = async () => {
  /**
   * In v3, the clients expose paginateOperationName APIs that are written using
 async generators so that you can use async iterators in a for await..of loop.
   * https://docs.aws.amazon.com/AWSJavaScriptSDK/v3/latest/index.html#paginators
   */
  const paginator = paginateListPolicies(
    { client, pageSize: 10 },
    // List only customer managed policies.
    { Scope: "Local" },
  );

  console.log("IAM policies defined in your account:");
  let policyCount = 0;
  for await (const page of paginator) {
    if (page.Policies) {
      page.Policies.forEach((p) => {
        console.log(`${p.PolicyName}`);
        policyCount++;
      });
    }
  }
  console.log(`Found ${policyCount} policies.`);
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考ListPolicies中的。

主题

- 操作
- 场景

操作

将策略附加到角色

以下代码示例展示了如何将 IAM policy 添加到角色。

适用于 JavaScript (v3) 的软件开发工具包

Note

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

附加策略。

```
import { AttachRolePolicyCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} policyArn
 * @param {string} roleName
 */
export const attachRolePolicy = (policyArn, roleName) => {
  const command = new AttachRolePolicyCommand({
    PolicyArn: policyArn,
    RoleName: roleName,
  });

  return client.send(command);
};
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考AttachRolePolicy中的。

适用于 JavaScript (v2) 的软件开发工具包

Note

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
// Load the AWS SDK for Node.js
```

```
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var paramsRoleList = {
  RoleName: process.argv[2],
};

iam.listAttachedRolePolicies(paramsRoleList, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    var myRolePolicies = data.AttachedPolicies;
    myRolePolicies.forEach(function (val, index, array) {
      if (myRolePolicies[index].PolicyName === "AmazonDynamoDBFullAccess") {
        console.log(
          "AmazonDynamoDBFullAccess is already attached to this role."
        );
        process.exit();
      }
    });
    var params = {
      PolicyArn: "arn:aws:iam::aws:policy/AmazonDynamoDBFullAccess",
      RoleName: process.argv[2],
    };
    iam.attachRolePolicy(params, function (err, data) {
      if (err) {
        console.log("Unable to attach policy to role", err);
      } else {
        console.log("Role attached successfully");
      }
    });
  }
});
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考AttachRolePolicy中的。

将内联策略附加到角色

以下代码示例展示了如何将内联策略附加到 IAM 角色。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { PutRolePolicyCommand, IAMClient } from "@aws-sdk/client-iam";

const examplePolicyDocument = JSON.stringify({
  Version: "2012-10-17",
  Statement: [
    {
      Sid: "VisualEditor0",
      Effect: "Allow",
      Action: [
        "s3:ListBucketMultipartUploads",
        "s3:ListBucketVersions",
        "s3:ListBucket",
        "s3:ListMultipartUploadParts",
      ],
      Resource: "arn:aws:s3:::some-test-bucket",
    },
    {
      Sid: "VisualEditor1",
      Effect: "Allow",
      Action: [
        "s3:ListStorageLensConfigurations",
        "s3:ListAccessPointsForObjectLambda",
        "s3:ListAllMyBuckets",
        "s3:ListAccessPoints",
        "s3:ListJobs",
        "s3:ListMultiRegionAccessPoints",
      ],
      Resource: "*",
    },
  ],
});
```

```
const client = new IAMClient({});

/**
 *
 * @param {string} roleName
 * @param {string} policyName
 * @param {string} policyDocument
 */
export const putRolePolicy = async (roleName, policyName, policyDocument) => {
  const command = new PutRolePolicyCommand({
    RoleName: roleName,
    PolicyName: policyName,
    PolicyDocument: policyDocument,
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考PutRolePolicy中的。

创建 SAML 提供者

以下代码示例显示了如何创建 Amazon Identity and Access Management (IAM) SAML 提供商。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { CreateSAMLProviderCommand, IAMClient } from "@aws-sdk/client-iam";
import { readFileSync } from "fs";
import * as path from "path";
import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";
```

```
const client = new IAMClient({});

/**
 * This sample document was generated using Auth0.
 * For more information on generating this document,
 see https://docs.aws.amazon.com/IAM/latest/UserGuide/
id_roles_providers_create_saml.html#samlstep1.
 */
const sampleMetadataDocument = readFileSync(
  path.join(
    dirnameFromMetaUrl(import.meta.url),
    "../../../../resources/sample_files/sample_saml_metadata.xml",
  ),
);

/**
 *
 * @param {*} providerName
 * @returns
 */
export const createSAMLProvider = async (providerName) => {
  const command = new CreateSAMLProviderCommand({
    Name: providerName,
    SAMLMetadataDocument: sampleMetadataDocument.toString(),
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- 有关 API 详细信息，请参阅《Amazon SDK for JavaScript API 参考》中的 CreateSAMLProvider。

创建组

以下代码示例展示了如何创建 IAM 组。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { CreateGroupCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} groupName
 */
export const createGroup = async (groupName) => {
  const command = new CreateGroupCommand({ GroupName: groupName });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考CreateGroup中的。

创建策略

以下代码示例展示了如何创建 IAM policy。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

创建 策略。

```
import { CreatePolicyCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} policyName
 */
export const createPolicy = (policyName) => {
  const command = new CreatePolicyCommand({
    PolicyDocument: JSON.stringify({
      Version: "2012-10-17",
      Statement: [
        {
          Effect: "Allow",
          Action: "*",
          Resource: "*",
        },
      ],
    }),
    PolicyName: policyName,
  });

  return client.send(command);
};
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考CreatePolicy中的。

适用于 JavaScript (v2) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });
```

```
// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var myManagedPolicy = {
  Version: "2012-10-17",
  Statement: [
    {
      Effect: "Allow",
      Action: "logs:CreateLogGroup",
      Resource: "RESOURCE_ARN",
    },
    {
      Effect: "Allow",
      Action: [
        "dynamodb:DeleteItem",
        "dynamodb:GetItem",
        "dynamodb:PutItem",
        "dynamodb:Scan",
        "dynamodb:UpdateItem",
      ],
      Resource: "RESOURCE_ARN",
    },
  ],
};

var params = {
  PolicyDocument: JSON.stringify(myManagedPolicy),
  PolicyName: "myDynamoDBPolicy",
};

iam.createPolicy(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考CreatePolicy中的。

创建角色

以下代码示例展示了如何创建 IAM 角色。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

创建角色。

```
import { CreateRoleCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} roleName
 */
export const createRole = (roleName) => {
  const command = new CreateRoleCommand({
    AssumeRolePolicyDocument: JSON.stringify({
      Version: "2012-10-17",
      Statement: [
        {
          Effect: "Allow",
          Principal: {
            Service: "lambda.amazonaws.com",
          },
          Action: "sts:AssumeRole",
        },
      ],
    }),
    RoleName: roleName,
  });

  return client.send(command);
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考CreateRole中的。

创建服务相关角色

以下代码示例展示了如何创建 IAM 服务相关角色。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

创建服务相关角色。

```
import { CreateServiceLinkedRoleCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} serviceName
 */
export const createServiceLinkedRole = async (serviceName) => {
  const command = new CreateServiceLinkedRoleCommand({
    // For a list of AWS services that support service-linked roles,
    // see https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_aws-services-
that-work-with-iam.html.
    //
    // For a list of AWS service endpoints, see https://docs.aws.amazon.com/general/
latest/gr/aws-service-information.html.
    AWSServiceName: serviceName,
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考CreateServiceLinkedRole中的。

创建用户

以下代码示例展示了如何创建 IAM 用户。

> **Warning**
>
> 为了避免安全风险，在开发专用软件或处理真实数据时，请勿使用 IAM 用户进行身份验证，而是使用与身份提供商的联合身份验证，例如 Amazon IAM Identity Center。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

创建 用户。

```
import { CreateUserCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} name
 */
export const createUser = (name) => {
  const command = new CreateUserCommand({ UserName: name });
  return client.send(command);
};
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考CreateUser中的。

适用于 JavaScript (v2) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  UserName: process.argv[2],
};

iam.getUser(params, function (err, data) {
  if (err && err.code === "NoSuchEntity") {
    iam.createUser(params, function (err, data) {
      if (err) {
        console.log("Error", err);
      } else {
        console.log("Success", data);
      }
    });
  } else {
    console.log(
      "User " + process.argv[2] + " already exists",
      data.User.UserId
    );
  }
});
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考CreateUser中的。

创建访问密钥

以下代码示例展示了如何创建 IAM 访问密钥。

> **Warning**
> 为了避免安全风险，在开发专用软件或处理真实数据时，请勿使用 IAM 用户进行身份验证，而是使用与身份提供商的联合身份验证，例如 Amazon IAM Identity Center。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

创建访问密钥。

```
import { CreateAccessKeyCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} userName
 */
export const createAccessKey = (userName) => {
  const command = new CreateAccessKeyCommand({ UserName: userName });
  return client.send(command);
};
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考CreateAccessKey中的。

适用于 JavaScript (v2) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

iam.createAccessKey({ UserName: "IAM_USER_NAME" }, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.AccessKey);
  }
});
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考CreateAccessKey中的。

为账户创建别名

以下代码示例展示了如何为 IAM 账户创建别名。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

创建账户别名。

```
import { CreateAccountAliasCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} alias - A unique name for the account alias.
 * @returns
 */
export const createAccountAlias = (alias) => {
  const command = new CreateAccountAliasCommand({
    AccountAlias: alias,
  });

  return client.send(command);
};
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考CreateAccountAlias中的。

适用于 JavaScript (v2) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

iam.createAccountAlias({ AccountAlias: process.argv[2] }, function (err, data) {
  if (err) {
    console.log("Error", err);
```

```
  } else {
    console.log("Success", data);
  }
});
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考CreateAccountAlias中的。

创建实例配置文件

以下代码示例演示了如何创建 IAM 实例配置文件。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
const { InstanceProfile } = await iamClient.send(
  new CreateInstanceProfileCommand({
    InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
  }),
);
await waitUntilInstanceProfileExists(
  { client: iamClient },
  { InstanceProfileName: NAMES.ssmOnlyInstanceProfileName },
);
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考CreateInstanceProfile中的。

删除 SAML 提供者

以下代码示例显示如何删除 Amazon Identity and Access Management (IAM) SAML 提供商。

适用于 JavaScript (v3) 的软件开发工具包

> Note
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { DeleteSAMLProviderCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} providerArn
 * @returns
 */
export const deleteSAMLProvider = async (providerArn) => {
  const command = new DeleteSAMLProviderCommand({
    SAMLProviderArn: providerArn,
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- 有关 API 详细信息，请参阅《Amazon SDK for JavaScript API 参考》中的 DeleteSAMLProvider。

删除组

以下代码示例展示了如何删除 IAM 组。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { DeleteGroupCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} groupName
 */
export const deleteGroup = async (groupName) => {
  const command = new DeleteGroupCommand({
    GroupName: groupName,
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DeleteGroup中的。

删除策略

以下代码示例展示了如何删除 IAM 策略。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

删除策略。

```
import { DeletePolicyCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} policyArn
 */
export const deletePolicy = (policyArn) => {
  const command = new DeletePolicyCommand({ PolicyArn: policyArn });
  return client.send(command);
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DeletePolicy中的。

删除角色

以下代码示例展示了如何删除 IAM 角色。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

删除角色。

```
import { DeleteRoleCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} roleName
 */
export const deleteRole = (roleName) => {
  const command = new DeleteRoleCommand({ RoleName: roleName });
```

```
  return client.send(command);
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DeleteRole中的。

删除角色策略

以下代码示例展示了如何删除 IAM 角色策略。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { DeleteRolePolicyCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} roleName
 * @param {string} policyName
 */
export const deleteRolePolicy = (roleName, policyName) => {
  const command = new DeleteRolePolicyCommand({
    RoleName: roleName,
    PolicyName: policyName,
  });
  return client.send(command);
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DeleteRolePolicy中的。

删除服务器证书

以下代码示例展示了如何删除 IAM 服务器证书。

适用于 JavaScript (v3) 的软件开发工具包

> Note
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

删除服务器证书。

```
import { DeleteServerCertificateCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} certName
 */
export const deleteServerCertificate = (certName) => {
  const command = new DeleteServerCertificateCommand({
    ServerCertificateName: certName,
  });

  return client.send(command);
};
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DeleteServerCertificate中的。

适用于 JavaScript (v2) 的软件开发工具包

> Note
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
// Load the AWS SDK for Node.js
```

```
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

iam.deleteServerCertificate(
  { ServerCertificateName: "CERTIFICATE_NAME" },
  function (err, data) {
    if (err) {
      console.log("Error", err);
    } else {
      console.log("Success", data);
    }
  }
);
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DeleteServerCertificate中的。

删除服务相关角色

以下代码示例展示了如何删除 IAM 服务相关角色。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { DeleteServiceLinkedRoleCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
```

```
 * @param {string} roleName
 */
export const deleteServiceLinkedRole = (roleName) => {
  const command = new DeleteServiceLinkedRoleCommand({ RoleName: roleName });
  return client.send(command);
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DeleteServiceLinkedRole中的。

删除用户

以下代码示例展示了如何删除 IAM 用户。

> **Warning**
>
> 为了避免安全风险，在开发专用软件或处理真实数据时，请勿使用 IAM 用户进行身份验证，而是使用与身份提供商的联合身份验证，例如 Amazon IAM Identity Center。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

删除用户。

```
import { DeleteUserCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} name
 */
export const deleteUser = (name) => {
```

```
  const command = new DeleteUserCommand({ UserName: name });
  return client.send(command);
};
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DeleteUser中的。

适用于 JavaScript (v2) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  UserName: process.argv[2],
};

iam.getUser(params, function (err, data) {
  if (err && err.code === "NoSuchEntity") {
    console.log("User " + process.argv[2] + " does not exist.");
  } else {
    iam.deleteUser(params, function (err, data) {
      if (err) {
        console.log("Error", err);
      } else {
        console.log("Success", data);
      }
    });
  }
});
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DeleteUser中的。

删除访问密钥

以下代码示例展示了如何删除 IAM 访问密钥。

Warning

为了避免安全风险，在开发专用软件或处理真实数据时，请勿使用 IAM 用户进行身份验证，而是使用与身份提供商的联合身份验证，例如 Amazon IAM Identity Center。

适用于 JavaScript (v3) 的软件开发工具包

Note

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

删除访问密钥。

```
import { DeleteAccessKeyCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} userName
 * @param {string} accessKeyId
 */
export const deleteAccessKey = (userName, accessKeyId) => {
  const command = new DeleteAccessKeyCommand({
    AccessKeyId: accessKeyId,
    UserName: userName,
  });

  return client.send(command);
};
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DeleteAccessKey中的。

适用于 JavaScript (v2) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  AccessKeyId: "ACCESS_KEY_ID",
  UserName: "USER_NAME",
};

iam.deleteAccessKey(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DeleteAccessKey中的。

删除账户别名

以下代码示例展示了如何删除 IAM 账户别名。

适用于 JavaScript (v3) 的软件开发工具包

Note

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

删除账户别名。

```
import { DeleteAccountAliasCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} alias
 */
export const deleteAccountAlias = (alias) => {
  const command = new DeleteAccountAliasCommand({ AccountAlias: alias });

  return client.send(command);
};
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DeleteAccountAlias中的。

适用于 JavaScript (v2) 的软件开发工具包

Note

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });
```

```
// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

iam.deleteAccountAlias({ AccountAlias: process.argv[2] }, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DeleteAccountAlias中的。

删除实例配置文件

以下代码示例演示了如何删除 IAM 实例配置文件。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
      const client = new IAMClient({});
      await client.send(
        new DeleteInstanceProfileCommand({
          InstanceProfileName: NAMES.instanceProfileName,
        }),
      );
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DeleteInstanceProfile中的。

从角色分离策略

以下代码示例展示了如何从角色分离 IAM 策略。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

分离策略。

```
import { DetachRolePolicyCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} policyArn
 * @param {string} roleName
 */
export const detachRolePolicy = (policyArn, roleName) => {
  const command = new DetachRolePolicyCommand({
    PolicyArn: policyArn,
    RoleName: roleName,
  });

  return client.send(command);
};
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DetachRolePolicy中的。

适用于 JavaScript (v2) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
// Load the AWS SDK for Node.js
```

```
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var paramsRoleList = {
  RoleName: process.argv[2],
};

iam.listAttachedRolePolicies(paramsRoleList, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    var myRolePolicies = data.AttachedPolicies;
    myRolePolicies.forEach(function (val, index, array) {
      if (myRolePolicies[index].PolicyName === "AmazonDynamoDBFullAccess") {
        var params = {
          PolicyArn: "arn:aws:iam::aws:policy/AmazonDynamoDBFullAccess",
          RoleName: process.argv[2],
        };
        iam.detachRolePolicy(params, function (err, data) {
          if (err) {
            console.log("Unable to detach policy from role", err);
          } else {
            console.log("Policy detached from role successfully");
            process.exit();
          }
        });
      }
    });
  }
});
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DetachRolePolicy中的。

获取策略

以下代码示例展示了如何获取 IAM 策略。

适用于 JavaScript (v3) 的软件开发工具包

> Note
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

获取策略。

```
import { GetPolicyCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} policyArn
 */
export const getPolicy = (policyArn) => {
  const command = new GetPolicyCommand({
    PolicyArn: policyArn,
  });

  return client.send(command);
};
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考GetPolicy中的。

适用于 JavaScript (v2) 的软件开发工具包

> Note
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
```

```
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  PolicyArn: "arn:aws:iam::aws:policy/AWSLambdaExecute",
};

iam.getPolicy(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Policy.Description);
  }
});
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考GetPolicy中的。

获取角色

以下代码示例展示了如何获取 IAM 角色。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

获取角色。

```
import { GetRoleCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
```

```
 *
 * @param {string} roleName
 */
export const getRole = (roleName) => {
  const command = new GetRoleCommand({
    RoleName: roleName,
  });

  return client.send(command);
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考GetRole中的。

获取服务器证书

以下代码示例展示了如何获取 IAM 服务器证书。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

获取服务器证书。

```
import { GetServerCertificateCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} certName
 * @returns
 */
export const getServerCertificate = async (certName) => {
  const command = new GetServerCertificateCommand({
    ServerCertificateName: certName,
  });
```

```
  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考GetServerCertificate中的。

适用于 JavaScript (v2) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

iam.getServerCertificate(
  { ServerCertificateName: "CERTIFICATE_NAME" },
  function (err, data) {
    if (err) {
      console.log("Error", err);
    } else {
      console.log("Success", data);
    }
  }
);
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考GetServerCertificate中的。

获取服务相关角色的删除状态

以下代码示例显示如何获取 Amazon Identity and Access Management (IAM) 服务相关角色的删除状态。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import {
  GetServiceLinkedRoleDeletionStatusCommand,
  IAMClient,
} from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} deletionTaskId
 */
export const getServiceLinkedRoleDeletionStatus = (deletionTaskId) => {
  const command = new GetServiceLinkedRoleDeletionStatusCommand({
    DeletionTaskId: deletionTaskId,
  });

  return client.send(command);
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考GetServiceLinkedRoleDeletionStatus中的。

获取有关上次使用访问密钥的数据

以下代码示例展示了如何获取关于上次使用 IAM 访问密钥的数据。

Warning

为了避免安全风险，在开发专用软件或处理真实数据时，请勿使用 IAM 用户进行身份验证，而是使用与身份提供商的联合身份验证，例如 Amazon IAM Identity Center。

适用于 JavaScript (v3) 的软件开发工具包

Note

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

获取访问密钥。

```
import { GetAccessKeyLastUsedCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} accessKeyId
 */
export const getAccessKeyLastUsed = async (accessKeyId) => {
  const command = new GetAccessKeyLastUsedCommand({
    AccessKeyId: accessKeyId,
  });

  const response = await client.send(command);

  if (response.AccessKeyLastUsed?.LastUsedDate) {
    console.log(`
    ${accessKeyId} was last used by ${response.UserName} via
    the ${response.AccessKeyLastUsed.ServiceName} service on
    ${response.AccessKeyLastUsed.LastUsedDate.toISOString()}
    `);
  }

  return response;
};
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考GetAccessKeyLastUsed中的。

适用于 JavaScript (v2) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

iam.getAccessKeyLastUsed(
  { AccessKeyId: "ACCESS_KEY_ID" },
  function (err, data) {
    if (err) {
      console.log("Error", err);
    } else {
      console.log("Success", data.AccessKeyLastUsed);
    }
  }
);
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考GetAccessKeyLastUsed中的。

获取账户密码策略

以下代码示例展示了如何获取 IAM 账户密码策略。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

获取账户密码策略。

```
import {
  GetAccountPasswordPolicyCommand,
  IAMClient,
} from "@aws-sdk/client-iam";

const client = new IAMClient({});

export const getAccountPasswordPolicy = async () => {
  const command = new GetAccountPasswordPolicyCommand({});

  const response = await client.send(command);
  console.log(response.PasswordPolicy);
  return response;
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考GetAccountPasswordPolicy中的。

列出 SAML 提供商

以下代码示例展示了如何列出 IAM 的 SAML 提供商。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

列出 SAML 提供商。

```
import { ListSAMLProvidersCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

export const listSamlProviders = async () => {
  const command = new ListSAMLProvidersCommand({});

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- 有关 API 详细信息，请参阅《Amazon SDK for JavaScript API 参考》中的 ListSAMLProviders。

列出用户的访问密钥

以下代码示例展示了如何列出用户的 IAM 访问密钥。

> Warning
>
> 为了避免安全风险，在开发专用软件或处理真实数据时，请勿使用 IAM 用户进行身份验证，而是使用与身份提供商的联合身份验证，例如 Amazon IAM Identity Center。

适用于 JavaScript (v3) 的软件开发工具包

> Note
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

列出访问密钥。

```
import { ListAccessKeysCommand, IAMClient } from "@aws-sdk/client-iam";
```

```
const client = new IAMClient({});

/**
 * A generator function that handles paginated results.
 * The AWS SDK for JavaScript (v3) provides {@link https://docs.aws.amazon.com/
AWSJavaScriptSDK/v3/latest/index.html#paginators | paginator} functions to simplify
 this.
 *
 * @param {string} userName
 */
export async function* listAccessKeys(userName) {
  const command = new ListAccessKeysCommand({
    MaxItems: 5,
    UserName: userName,
  });

  /**
   * @type {import("@aws-sdk/client-iam").ListAccessKeysCommandOutput | undefined}
   */
  let response = await client.send(command);

  while (response?.AccessKeyMetadata?.length) {
    for (const key of response.AccessKeyMetadata) {
      yield key;
    }

    if (response.IsTruncated) {
      response = await client.send(
        new ListAccessKeysCommand({
          Marker: response.Marker,
        }),
      );
    } else {
      break;
    }
  }
}
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考ListAccessKeys中的。

适用于 JavaScript (v2) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  MaxItems: 5,
  UserName: "IAM_USER_NAME",
};

iam.listAccessKeys(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考ListAccessKeys中的。

列出账户别名

以下代码示例展示了如何列出 IAM 账户别名。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

列出账户别名。

```
import { ListAccountAliasesCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 * A generator function that handles paginated results.
 * The AWS SDK for JavaScript (v3) provides {@link https://docs.aws.amazon.com/
AWSJavaScriptSDK/v3/latest/index.html#paginators | paginator} functions to simplify
 this.
 */
export async function* listAccountAliases() {
  const command = new ListAccountAliasesCommand({ MaxItems: 5 });

  let response = await client.send(command);

  while (response.AccountAliases?.length) {
    for (const alias of response.AccountAliases) {
      yield alias;
    }

    if (response.IsTruncated) {
      response = await client.send(
        new ListAccountAliasesCommand({
          Marker: response.Marker,
          MaxItems: 5,
        }),
      );
    } else {
      break;
    }
  }
}
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考ListAccountAliases中的。

适用于 JavaScript (v2) 的软件开发工具包

> **Note**
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

iam.listAccountAliases({ MaxItems: 10 }, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考ListAccountAliases中的。

列出组

以下代码示例展示了如何列出 IAM 组。

适用于 JavaScript (v3) 的软件开发工具包

Note

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

列出组。

```
import { ListGroupsCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 * A generator function that handles paginated results.
 * The AWS SDK for JavaScript (v3) provides {@link https://docs.aws.amazon.com/
AWSJavaScriptSDK/v3/latest/index.html#paginators | paginator} functions to simplify
 this.
 */
export async function* listGroups() {
  const command = new ListGroupsCommand({
    MaxItems: 10,
  });

  let response = await client.send(command);

  while (response.Groups?.length) {
    for (const group of response.Groups) {
      yield group;
    }

    if (response.IsTruncated) {
      response = await client.send(
        new ListGroupsCommand({
          Marker: response.Marker,
          MaxItems: 10,
        }),
      );
    } else {
      break;
    }
  }
```

```
}
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考ListGroups中的。

列出角色的内联策略

以下代码示例展示了如何列出 IAM 角色的内联策略。

适用于 JavaScript (v3) 的软件开发工具包

> Note
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

列出策略。

```
import { ListRolePoliciesCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 * A generator function that handles paginated results.
 * The AWS SDK for JavaScript (v3) provides {@link https://docs.aws.amazon.com/
AWSJavaScriptSDK/v3/latest/index.html#paginators | paginator} functions to simplify
 this.
 *
 * @param {string} roleName
 */
export async function* listRolePolicies(roleName) {
  const command = new ListRolePoliciesCommand({
    RoleName: roleName,
    MaxItems: 10,
  });

  let response = await client.send(command);

  while (response.PolicyNames?.length) {
    for (const policyName of response.PolicyNames) {
      yield policyName;
```

```
    }

    if (response.IsTruncated) {
      response = await client.send(
        new ListRolePoliciesCommand({
          RoleName: roleName,
          MaxItems: 10,
          Marker: response.Marker,
        }),
      );
    } else {
      break;
    }
  }
}
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考ListRolePolicies中的。

列出策略

以下代码示例展示了如何列出 IAM 策略。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

列出策略。

```
import { ListPoliciesCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 * A generator function that handles paginated results.
 * The AWS SDK for JavaScript (v3) provides {@link https://docs.aws.amazon.com/
AWSJavaScriptSDK/v3/latest/index.html#paginators | paginator} functions to simplify
 this.
```

```
 *
 */
export async function* listPolicies() {
  const command = new ListPoliciesCommand({
    MaxItems: 10,
    OnlyAttached: false,
    // List only the customer managed policies in your Amazon Web Services account.
    Scope: "Local",
  });

  let response = await client.send(command);

  while (response.Policies?.length) {
    for (const policy of response.Policies) {
      yield policy;
    }

    if (response.IsTruncated) {
      response = await client.send(
        new ListPoliciesCommand({
          Marker: response.Marker,
          MaxItems: 10,
          OnlyAttached: false,
          Scope: "Local",
        }),
      );
    } else {
      break;
    }
  }
}
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考ListPolicies中的。

列出附加到角色的策略

以下代码示例展示了如何列出附加到 IAM 角色的策略。

适用于 JavaScript (v3) 的软件开发工具包

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

列出附加到角色的策略。

```
import {
  ListAttachedRolePoliciesCommand,
  IAMClient,
} from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 * A generator function that handles paginated results.
 * The AWS SDK for JavaScript (v3) provides {@link https://docs.aws.amazon.com/
AWSJavaScriptSDK/v3/latest/index.html#paginators | paginator} functions to simplify
 this.
 * @param {string} roleName
 */
export async function* listAttachedRolePolicies(roleName) {
  const command = new ListAttachedRolePoliciesCommand({
    RoleName: roleName,
  });

  let response = await client.send(command);

  while (response.AttachedPolicies?.length) {
    for (const policy of response.AttachedPolicies) {
      yield policy;
    }

    if (response.IsTruncated) {
      response = await client.send(
        new ListAttachedRolePoliciesCommand({
          RoleName: roleName,
          Marker: response.Marker,
        }),
      );
```

```
    } else {
      break;
    }
  }
}
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考ListAttachedRolePolicies中的。

列出角色

以下代码示例展示了如何列出 IAM 角色。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

列出角色。

```
import { ListRolesCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 * A generator function that handles paginated results.
 * The AWS SDK for JavaScript (v3) provides {@link https://docs.aws.amazon.com/
AWSJavaScriptSDK/v3/latest/index.html#paginators | paginator} functions to simplify
 this.
 *
 */
export async function* listRoles() {
  const command = new ListRolesCommand({
    MaxItems: 10,
  });

  /**
```

```
   * @type {import("@aws-sdk/client-iam").ListRolesCommandOutput | undefined}
   */
  let response = await client.send(command);

  while (response?.Roles?.length) {
    for (const role of response.Roles) {
      yield role;
    }

    if (response.IsTruncated) {
      response = await client.send(
        new ListRolesCommand({
          Marker: response.Marker,
        }),
      );
    } else {
      break;
    }
  }
}
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考ListRoles中的。

列出服务器证书

以下代码示例展示了如何列出 IAM 服务器证书。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

列出证书。

```
import { ListServerCertificatesCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});
```

```
/**
 * A generator function that handles paginated results.
 * The AWS SDK for JavaScript (v3) provides {@link https://docs.aws.amazon.com/
AWSJavaScriptSDK/v3/latest/index.html#paginators | paginator} functions to simplify
 this.
 *
 */
export async function* listServerCertificates() {
  const command = new ListServerCertificatesCommand({});
  let response = await client.send(command);

  while (response.ServerCertificateMetadataList?.length) {
    for await (const cert of response.ServerCertificateMetadataList) {
      yield cert;
    }

    if (response.IsTruncated) {
      response = await client.send(new ListServerCertificatesCommand({}));
    } else {
      break;
    }
  }
}
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考ListServerCertificates中的。

适用于 JavaScript (v2) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
```

```
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

iam.listServerCertificates({}, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考ListServerCertificates中的。

列出用户

以下代码示例展示了如何列出 IAM 用户。

> **Warning**
>
> 为了避免安全风险，在开发专用软件或处理真实数据时，请勿使用 IAM 用户进行身份验证，而是使用与身份提供商的联合身份验证，例如 Amazon IAM Identity Center。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

列出用户。

```
import { ListUsersCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

export const listUsers = async () => {
```

```
  const command = new ListUsersCommand({ MaxItems: 10 });

  const response = await client.send(command);
  response.Users?.forEach(({ UserName, CreateDate }) => {
    console.log(`${UserName} created on: ${CreateDate}`);
  });
  return response;
};
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考ListUsers中的。

适用于 JavaScript (v2) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  MaxItems: 10,
};

iam.listUsers(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    var users = data.Users || [];
    users.forEach(function (user) {
      console.log("User " + user.UserName + " created", user.CreateDate);
    });
  }
```

```
});
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考ListUsers中的。

更新服务器证书

以下代码示例展示了如何更新 IAM 服务器证书。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

更新服务器证书。

```
import { UpdateServerCertificateCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} currentName
 * @param {string} newName
 */
export const updateServerCertificate = (currentName, newName) => {
  const command = new UpdateServerCertificateCommand({
    ServerCertificateName: currentName,
    NewServerCertificateName: newName,
  });

  return client.send(command);
};
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考UpdateServerCertificate中的。

适用于 JavaScript (v2) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  ServerCertificateName: "CERTIFICATE_NAME",
  NewServerCertificateName: "NEW_CERTIFICATE_NAME",
};

iam.updateServerCertificate(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考UpdateServerCertificate中的。

更新用户

以下代码示例展示了如何更新 IAM 用户。

Warning

为了避免安全风险，在开发专用软件或处理真实数据时，请勿使用 IAM 用户进行身份验证，而是使用与身份提供商的联合身份验证，例如 Amazon IAM Identity Center。

适用于 JavaScript (v3) 的软件开发工具包

Note

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

更新用户。

```
import { UpdateUserCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} currentUserName
 * @param {string} newUserName
 */
export const updateUser = (currentUserName, newUserName) => {
  const command = new UpdateUserCommand({
    UserName: currentUserName,
    NewUserName: newUserName,
  });

  return client.send(command);
};
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考UpdateUser中的。

适用于 JavaScript (v2) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  UserName: process.argv[2],
  NewUserName: process.argv[3],
};

iam.updateUser(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考UpdateUser中的。

更新访问密钥

以下代码示例展示了如何更新 IAM 访问密钥。

Warning

为了避免安全风险，在开发专用软件或处理真实数据时，请勿使用 IAM 用户进行身份验证，而是使用与身份提供商的联合身份验证，例如 Amazon IAM Identity Center。

适用于 JavaScript (v3) 的软件开发工具包

Note

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

更新访问密钥。

```
import {
  UpdateAccessKeyCommand,
  IAMClient,
  StatusType,
} from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} userName
 * @param {string} accessKeyId
 */
export const updateAccessKey = (userName, accessKeyId) => {
  const command = new UpdateAccessKeyCommand({
    AccessKeyId: accessKeyId,
    Status: StatusType.Inactive,
    UserName: userName,
  });

  return client.send(command);
};
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考UpdateAccessKey中的。

适用于 JavaScript (v2) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  AccessKeyId: "ACCESS_KEY_ID",
  Status: "Active",
  UserName: "USER_NAME",
};

iam.updateAccessKey(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考UpdateAccessKey中的。

上传服务器证书

以下代码示例显示如何上传 Amazon Identity and Access Management (IAM) 服务器证书。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { UploadServerCertificateCommand, IAMClient } from "@aws-sdk/client-iam";
import { readFileSync } from "fs";
import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";
import * as path from "path";

const client = new IAMClient({});

const certMessage = `Generate a certificate and key with the following command, or
 the equivalent for your system.

openssl req -x509 -newkey rsa:4096 -sha256 -days 3650 -nodes \
-keyout example.key -out example.crt -subj "/CN=example.com" \
-addext "subjectAltName=DNS:example.com,DNS:www.example.net,IP:10.0.0.1"
`;

const getCertAndKey = () => {
  try {
    const cert = readFileSync(
      path.join(dirnameFromMetaUrl(import.meta.url), "./example.crt"),
    );
    const key = readFileSync(
      path.join(dirnameFromMetaUrl(import.meta.url), "./example.key"),
    );
    return { cert, key };
  } catch (err) {
    if (err.code === "ENOENT") {
      throw new Error(
        `Certificate and/or private key not found. ${certMessage}`,
      );
    }

    throw err;
  }
};
```

```
/**
 *
 * @param {string} certificateName
 */
export const uploadServerCertificate = (certificateName) => {
  const { cert, key } = getCertAndKey();
  const command = new UploadServerCertificateCommand({
    ServerCertificateName: certificateName,
    CertificateBody: cert.toString(),
    PrivateKey: key.toString(),
  });

  return client.send(command);
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考UploadServerCertificate中的。

场景

构建和管理弹性服务

以下代码示例演示了如何创建可返回书籍、电影和歌曲推荐的负载均衡的 Web 服务。该示例演示服务如何响应故障，以及如何重组服务以提高故障发生时的弹性。

- 使用 Amazon EC2 Auto Scaling 组根据启动模板创建 Amazon Elastic Compute Cloud（Amazon EC2）实例，并将实例数量保持在指定范围内。
- 使用弹性负载均衡处理和分发 HTTP 请求。
- 监控自动扩缩组中实例的运行状况，并仅将请求转发到运行状况良好的实例。
- 在每个 EC2 实例上运行 Python Web 服务器以处理 HTTP 请求。Web 服务器以建议和运行状况检查作为响应。
- 使用 Amazon DynamoDB 表模拟推荐服务。
- 通过更新 Amazon Systems Manager 参数来控制 Web 服务器对请求和运行状况检查的响应。

适用于 JavaScript (v3) 的软件开发工具包

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

在命令提示符中运行交互式场景。

```
#!/usr/bin/env node
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0


import {
  Scenario,
  parseScenarioArgs,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

/**
 * The workflow steps are split into three stages:
 *   - deploy
 *   - demo
 *   - destroy
 *
 * Each of these stages has a corresponding file prefixed with steps-*.
 */
import { deploySteps } from "./steps-deploy.js";
import { demoSteps } from "./steps-demo.js";
import { destroySteps } from "./steps-destroy.js";

/**
 * The context is passed to every scenario. Scenario steps
 * will modify the context.
 */
const context = {};

/**
 * Three Scenarios are created for the workflow. A Scenario is an orchestration
 class
 * that simplifies running a series of steps.
 */
```

```
export const scenarios = {
  // Deploys all resources necessary for the workflow.
  deploy: new Scenario("Resilient Workflow - Deploy", deploySteps, context),
  // Demonstrates how a fragile web service can be made more resilient.
  demo: new Scenario("Resilient Workflow - Demo", demoSteps, context),
  // Destroys the resources created for the workflow.
  destroy: new Scenario("Resilient Workflow - Destroy", destroySteps, context),
};

// Call function if run directly
import { fileURLToPath } from "url";

if (process.argv[1] === fileURLToPath(import.meta.url)) {
  parseScenarioArgs(scenarios);
}
```

创建部署所有资源的步骤。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { join } from "node:path";
import { readFileSync, writeFileSync } from "node:fs";
import axios from "axios";

import {
  BatchWriteItemCommand,
  CreateTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";
import {
  EC2Client,
  CreateKeyPairCommand,
  CreateLaunchTemplateCommand,
  DescribeAvailabilityZonesCommand,
  DescribeVpcsCommand,
  DescribeSubnetsCommand,
  DescribeSecurityGroupsCommand,
  AuthorizeSecurityGroupIngressCommand,
} from "@aws-sdk/client-ec2";
import {
  IAMClient,
```

```
  CreatePolicyCommand,
  CreateRoleCommand,
  CreateInstanceProfileCommand,
  AddRoleToInstanceProfileCommand,
  AttachRolePolicyCommand,
  waitUntilInstanceProfileExists,
} from "@aws-sdk/client-iam";
import { SSMClient, GetParameterCommand } from "@aws-sdk/client-ssm";
import {
  CreateAutoScalingGroupCommand,
  AutoScalingClient,
  AttachLoadBalancerTargetGroupsCommand,
} from "@aws-sdk/client-auto-scaling";
import {
  CreateListenerCommand,
  CreateLoadBalancerCommand,
  CreateTargetGroupCommand,
  ElasticLoadBalancingV2Client,
  waitUntilLoadBalancerAvailable,
} from "@aws-sdk/client-elastic-load-balancing-v2";

import {
  ScenarioOutput,
  ScenarioInput,
  ScenarioAction,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES, RESOURCES_PATH, ROOT } from "./constants.js";
import { initParamsSteps } from "./steps-reset-params.js";

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const deploySteps = [
  new ScenarioOutput("introduction", MESSAGES.introduction, { header: true }),
  new ScenarioInput("confirmDeployment", MESSAGES.confirmDeployment, {
    type: "confirm",
  }),
  new ScenarioAction(
    "handleConfirmDeployment",
    (c) => c.confirmDeployment === false && process.exit(),
  ),
  new ScenarioOutput(
```

```
    "creatingTable",
    MESSAGES.creatingTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
  new ScenarioAction("createTable", async () => {
    const client = new DynamoDBClient({});
    await client.send(
      new CreateTableCommand({
        TableName: NAMES.tableName,
        ProvisionedThroughput: {
          ReadCapacityUnits: 5,
          WriteCapacityUnits: 5,
        },
        AttributeDefinitions: [
          {
            AttributeName: "MediaType",
            AttributeType: "S",
          },
          {
            AttributeName: "ItemId",
            AttributeType: "N",
          },
        ],
        KeySchema: [
          {
            AttributeName: "MediaType",
            KeyType: "HASH",
          },
          {
            AttributeName: "ItemId",
            KeyType: "RANGE",
          },
        ],
      }),
    );
    await waitUntilTableExists({ client }, { TableName: NAMES.tableName });
  }),
  new ScenarioOutput(
    "createdTable",
    MESSAGES.createdTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
  new ScenarioOutput(
    "populatingTable",
    MESSAGES.populatingTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
```

```
  new ScenarioAction("populateTable", () => {
    const client = new DynamoDBClient({});
    /**
     * @type {{ default: import("@aws-sdk/client-dynamodb").PutRequest['Item'][] }}
     */
    const recommendations = JSON.parse(
      readFileSync(join(RESOURCES_PATH, "recommendations.json")),
    );

    return client.send(
      new BatchWriteItemCommand({
        RequestItems: {
          [NAMES.tableName]: recommendations.map((item) => ({
            PutRequest: { Item: item },
          })),
        },
      }),
    );
  }),
  new ScenarioOutput(
    "populatedTable",
    MESSAGES.populatedTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
  new ScenarioOutput(
    "creatingKeyPair",
    MESSAGES.creatingKeyPair.replace("${KEY_PAIR_NAME}", NAMES.keyPairName),
  ),
  new ScenarioAction("createKeyPair", async () => {
    const client = new EC2Client({});
    const { KeyMaterial } = await client.send(
      new CreateKeyPairCommand({
        KeyName: NAMES.keyPairName,
      }),
    );

    writeFileSync(`${NAMES.keyPairName}.pem`, KeyMaterial, { mode: 0o600 });
  }),
  new ScenarioOutput(
    "createdKeyPair",
    MESSAGES.createdKeyPair.replace("${KEY_PAIR_NAME}", NAMES.keyPairName),
  ),
  new ScenarioOutput(
    "creatingInstancePolicy",
    MESSAGES.creatingInstancePolicy.replace(
```

```
      "${INSTANCE_POLICY_NAME}",
      NAMES.instancePolicyName,
    ),
  ),
  new ScenarioAction("createInstancePolicy", async (state) => {
    const client = new IAMClient({});
    const {
      Policy: { Arn },
    } = await client.send(
      new CreatePolicyCommand({
        PolicyName: NAMES.instancePolicyName,
        PolicyDocument: readFileSync(
          join(RESOURCES_PATH, "instance_policy.json"),
        ),
      }),
    );
    state.instancePolicyArn = Arn;
  }),
  new ScenarioOutput("createdInstancePolicy", (state) =>
    MESSAGES.createdInstancePolicy
      .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
      .replace("${INSTANCE_POLICY_ARN}", state.instancePolicyArn),
  ),
  new ScenarioOutput(
    "creatingInstanceRole",
    MESSAGES.creatingInstanceRole.replace(
      "${INSTANCE_ROLE_NAME}",
      NAMES.instanceRoleName,
    ),
  ),
  new ScenarioAction("createInstanceRole", () => {
    const client = new IAMClient({});
    return client.send(
      new CreateRoleCommand({
        RoleName: NAMES.instanceRoleName,
        AssumeRolePolicyDocument: readFileSync(
          join(ROOT, "assume-role-policy.json"),
        ),
      }),
    );
  }),
  new ScenarioOutput(
    "createdInstanceRole",
    MESSAGES.createdInstanceRole.replace(
```

```
        "${INSTANCE_ROLE_NAME}",
        NAMES.instanceRoleName,
      ),
    ),
    new ScenarioOutput(
      "attachingPolicyToRole",
      MESSAGES.attachingPolicyToRole
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName)
        .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName),
    ),
    new ScenarioAction("attachPolicyToRole", async (state) => {
      const client = new IAMClient({});
      await client.send(
        new AttachRolePolicyCommand({
          RoleName: NAMES.instanceRoleName,
          PolicyArn: state.instancePolicyArn,
        }),
      );
    }),
    new ScenarioOutput(
      "attachedPolicyToRole",
      MESSAGES.attachedPolicyToRole
        .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
    ),
    new ScenarioOutput(
      "creatingInstanceProfile",
      MESSAGES.creatingInstanceProfile.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.instanceProfileName,
      ),
    ),
    new ScenarioAction("createInstanceProfile", async (state) => {
      const client = new IAMClient({});
      const {
        InstanceProfile: { Arn },
      } = await client.send(
        new CreateInstanceProfileCommand({
          InstanceProfileName: NAMES.instanceProfileName,
        }),
      );
      state.instanceProfileArn = Arn;

      await waitUntilInstanceProfileExists(
```

```
      { client },
      { InstanceProfileName: NAMES.instanceProfileName },
    );
  }),
  new ScenarioOutput("createdInstanceProfile", (state) =>
    MESSAGES.createdInstanceProfile
      .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
      .replace("${INSTANCE_PROFILE_ARN}", state.instanceProfileArn),
  ),
  new ScenarioOutput(
    "addingRoleToInstanceProfile",
    MESSAGES.addingRoleToInstanceProfile
      .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
  ),
  new ScenarioAction("addRoleToInstanceProfile", () => {
    const client = new IAMClient({});
    return client.send(
      new AddRoleToInstanceProfileCommand({
        RoleName: NAMES.instanceRoleName,
        InstanceProfileName: NAMES.instanceProfileName,
      }),
    );
  }),
  new ScenarioOutput(
    "addedRoleToInstanceProfile",
    MESSAGES.addedRoleToInstanceProfile
      .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
  ),
  ...initParamsSteps,
  new ScenarioOutput("creatingLaunchTemplate", MESSAGES.creatingLaunchTemplate),
  new ScenarioAction("createLaunchTemplate", async () => {
    // snippet-start:[javascript.v3.wkflw.resilient.CreateLaunchTemplate]
    const ssmClient = new SSMClient({});
    const { Parameter } = await ssmClient.send(
      new GetParameterCommand({
        Name: "/aws/service/ami-amazon-linux-latest/amzn2-ami-hvm-x86_64-gp2",
      }),
    );
    const ec2Client = new EC2Client({});
    await ec2Client.send(
      new CreateLaunchTemplateCommand({
        LaunchTemplateName: NAMES.launchTemplateName,
```

```
        LaunchTemplateData: {
          InstanceType: "t3.micro",
          ImageId: Parameter.Value,
          IamInstanceProfile: { Name: NAMES.instanceProfileName },
          UserData: readFileSync(
            join(RESOURCES_PATH, "server_startup_script.sh"),
          ).toString("base64"),
          KeyName: NAMES.keyPairName,
        },
      }),
      // snippet-end:[javascript.v3.wkflw.resilient.CreateLaunchTemplate]
    );
  }),
  new ScenarioOutput(
    "createdLaunchTemplate",
    MESSAGES.createdLaunchTemplate.replace(
      "${LAUNCH_TEMPLATE_NAME}",
      NAMES.launchTemplateName,
    ),
  ),
  new ScenarioOutput(
    "creatingAutoScalingGroup",
    MESSAGES.creatingAutoScalingGroup.replace(
      "${AUTO_SCALING_GROUP_NAME}",
      NAMES.autoScalingGroupName,
    ),
  ),
  new ScenarioAction("createAutoScalingGroup", async (state) => {
    const ec2Client = new EC2Client({});
    const { AvailabilityZones } = await ec2Client.send(
      new DescribeAvailabilityZonesCommand({}),
    );
    state.availabilityZoneNames = AvailabilityZones.map((az) => az.ZoneName);
    const autoScalingClient = new AutoScalingClient({});
    await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
      autoScalingClient.send(
        new CreateAutoScalingGroupCommand({
          AvailabilityZones: state.availabilityZoneNames,
          AutoScalingGroupName: NAMES.autoScalingGroupName,
          LaunchTemplate: {
            LaunchTemplateName: NAMES.launchTemplateName,
            Version: "$Default",
          },
          MinSize: 3,
```

```
          MaxSize: 3,
        }),
      ),
    );
  }),
  new ScenarioOutput(
    "createdAutoScalingGroup",
    /**
     * @param {{ availabilityZoneNames: string[] }} state
     */
    (state) =>
      MESSAGES.createdAutoScalingGroup
        .replace("${AUTO_SCALING_GROUP_NAME}", NAMES.autoScalingGroupName)
        .replace(
          "${AVAILABILITY_ZONE_NAMES}",
          state.availabilityZoneNames.join(", "),
        ),
  ),
  new ScenarioInput("confirmContinue", MESSAGES.confirmContinue, {
    type: "confirm",
  }),
  new ScenarioOutput("loadBalancer", MESSAGES.loadBalancer),
  new ScenarioOutput("gettingVpc", MESSAGES.gettingVpc),
  new ScenarioAction("getVpc", async (state) => {
    // snippet-start:[javascript.v3.wkflw.resilient.DescribeVpcs]
    const client = new EC2Client({});
    const { Vpcs } = await client.send(
      new DescribeVpcsCommand({
        Filters: [{ Name: "is-default", Values: ["true"] }],
      }),
    );
    // snippet-end:[javascript.v3.wkflw.resilient.DescribeVpcs]
    state.defaultVpc = Vpcs[0].VpcId;
  }),
  new ScenarioOutput("gotVpc", (state) =>
    MESSAGES.gotVpc.replace("${VPC_ID}", state.defaultVpc),
  ),
  new ScenarioOutput("gettingSubnets", MESSAGES.gettingSubnets),
  new ScenarioAction("getSubnets", async (state) => {
    // snippet-start:[javascript.v3.wkflw.resilient.DescribeSubnets]
    const client = new EC2Client({});
    const { Subnets } = await client.send(
      new DescribeSubnetsCommand({
        Filters: [
```

```
          { Name: "vpc-id", Values: [state.defaultVpc] },
          { Name: "availability-zone", Values: state.availabilityZoneNames },
          { Name: "default-for-az", Values: ["true"] },
        ],
      }),
    );
    // snippet-end:[javascript.v3.wkflw.resilient.DescribeSubnets]
    state.subnets = Subnets.map((subnet) => subnet.SubnetId);
  }),
  new ScenarioOutput(
    "gotSubnets",
    /**
     * @param {{ subnets: string[] }} state
     */
    (state) =>
      MESSAGES.gotSubnets.replace("${SUBNETS}", state.subnets.join(", ")),
  ),
  new ScenarioOutput(
    "creatingLoadBalancerTargetGroup",
    MESSAGES.creatingLoadBalancerTargetGroup.replace(
      "${TARGET_GROUP_NAME}",
      NAMES.loadBalancerTargetGroupName,
    ),
  ),
  new ScenarioAction("createLoadBalancerTargetGroup", async (state) => {
    // snippet-start:[javascript.v3.wkflw.resilient.CreateTargetGroup]
    const client = new ElasticLoadBalancingV2Client({});
    const { TargetGroups } = await client.send(
      new CreateTargetGroupCommand({
        Name: NAMES.loadBalancerTargetGroupName,
        Protocol: "HTTP",
        Port: 80,
        HealthCheckPath: "/healthcheck",
        HealthCheckIntervalSeconds: 10,
        HealthCheckTimeoutSeconds: 5,
        HealthyThresholdCount: 2,
        UnhealthyThresholdCount: 2,
        VpcId: state.defaultVpc,
      }),
    );
    // snippet-end:[javascript.v3.wkflw.resilient.CreateTargetGroup]
    const targetGroup = TargetGroups[0];
    state.targetGroupArn = targetGroup.TargetGroupArn;
    state.targetGroupProtocol = targetGroup.Protocol;
```

```
      state.targetGroupPort = targetGroup.Port;
    }),
    new ScenarioOutput(
      "createdLoadBalancerTargetGroup",
      MESSAGES.createdLoadBalancerTargetGroup.replace(
        "${TARGET_GROUP_NAME}",
        NAMES.loadBalancerTargetGroupName,
      ),
    ),
    new ScenarioOutput(
      "creatingLoadBalancer",
      MESSAGES.creatingLoadBalancer.replace("${LB_NAME}", NAMES.loadBalancerName),
    ),
    new ScenarioAction("createLoadBalancer", async (state) => {
      // snippet-start:[javascript.v3.wkflw.resilient.CreateLoadBalancer]
      const client = new ElasticLoadBalancingV2Client({});
      const { LoadBalancers } = await client.send(
        new CreateLoadBalancerCommand({
          Name: NAMES.loadBalancerName,
          Subnets: state.subnets,
        }),
      );
      state.loadBalancerDns = LoadBalancers[0].DNSName;
      state.loadBalancerArn = LoadBalancers[0].LoadBalancerArn;
      await waitUntilLoadBalancerAvailable(
        { client },
        { Names: [NAMES.loadBalancerName] },
      );
      // snippet-end:[javascript.v3.wkflw.resilient.CreateLoadBalancer]
    }),
    new ScenarioOutput("createdLoadBalancer", (state) =>
      MESSAGES.createdLoadBalancer
        .replace("${LB_NAME}", NAMES.loadBalancerName)
        .replace("${DNS_NAME}", state.loadBalancerDns),
    ),
    new ScenarioOutput(
      "creatingListener",
      MESSAGES.creatingLoadBalancerListener
        .replace("${LB_NAME}", NAMES.loadBalancerName)
        .replace("${TARGET_GROUP_NAME}", NAMES.loadBalancerTargetGroupName),
    ),
    new ScenarioAction("createListener", async (state) => {
      // snippet-start:[javascript.v3.wkflw.resilient.CreateListener]
      const client = new ElasticLoadBalancingV2Client({});
```

```
    const { Listeners } = await client.send(
      new CreateListenerCommand({
        LoadBalancerArn: state.loadBalancerArn,
        Protocol: state.targetGroupProtocol,
        Port: state.targetGroupPort,
        DefaultActions: [
          { Type: "forward", TargetGroupArn: state.targetGroupArn },
        ],
      }),
    );
    // snippet-end:[javascript.v3.wkflw.resilient.CreateListener]
    const listener = Listeners[0];
    state.loadBalancerListenerArn = listener.ListenerArn;
  }),
  new ScenarioOutput("createdListener", (state) =>
    MESSAGES.createdLoadBalancerListener.replace(
      "${LB_LISTENER_ARN}",
      state.loadBalancerListenerArn,
    ),
  ),
  new ScenarioOutput(
    "attachingLoadBalancerTargetGroup",
    MESSAGES.attachingLoadBalancerTargetGroup
      .replace("${TARGET_GROUP_NAME}", NAMES.loadBalancerTargetGroupName)
      .replace("${AUTO_SCALING_GROUP_NAME}", NAMES.autoScalingGroupName),
  ),
  new ScenarioAction("attachLoadBalancerTargetGroup", async (state) => {
    // snippet-start:[javascript.v3.wkflw.resilient.AttachTargetGroup]
    const client = new AutoScalingClient({});
    await client.send(
      new AttachLoadBalancerTargetGroupsCommand({
        AutoScalingGroupName: NAMES.autoScalingGroupName,
        TargetGroupARNs: [state.targetGroupArn],
      }),
    );
    // snippet-end:[javascript.v3.wkflw.resilient.AttachTargetGroup]
  }),
  new ScenarioOutput(
    "attachedLoadBalancerTargetGroup",
    MESSAGES.attachedLoadBalancerTargetGroup,
  ),
  new ScenarioOutput("verifyingInboundPort", MESSAGES.verifyingInboundPort),
  new ScenarioAction(
    "verifyInboundPort",
```

```
    /**
     *
     * @param {{ defaultSecurityGroup: import('@aws-sdk/client-ec2').SecurityGroup}}
  state
     */
    async (state) => {
      const client = new EC2Client({});
      const { SecurityGroups } = await client.send(
        new DescribeSecurityGroupsCommand({
          Filters: [{ Name: "group-name", Values: ["default"] }],
        }),
      );
      if (!SecurityGroups) {
        state.verifyInboundPortError = new Error(MESSAGES.noSecurityGroups);
      }
      state.defaultSecurityGroup = SecurityGroups[0];

      /**
       * @type {string}
       */
      const ipResponse = (await axios.get("http://checkip.amazonaws.com")).data;
      state.myIp = ipResponse.trim();
      const myIpRules = state.defaultSecurityGroup.IpPermissions.filter(
        ({ IpRanges }) =>
          IpRanges.some(
            ({ CidrIp }) =>
              CidrIp.startsWith(state.myIp) || CidrIp === "0.0.0.0/0",
          ),
      )
        .filter(({ IpProtocol }) => IpProtocol === "tcp")
        .filter(({ FromPort }) => FromPort === 80);

      state.myIpRules = myIpRules;
    },
  ),
  new ScenarioOutput(
    "verifiedInboundPort",
    /**
     * @param {{ myIpRules: any[] }} state
     */
    (state) => {
      if (state.myIpRules.length > 0) {
        return MESSAGES.foundIpRules.replace(
          "${IP_RULES}",
```

```
          JSON.stringify(state.myIpRules, null, 2),
        );
      } else {
        return MESSAGES.noIpRules;
      }
    },
  ),
  new ScenarioInput(
    "shouldAddInboundRule",
    /**
     * @param {{ myIpRules: any[] }} state
     */
    (state) => {
      if (state.myIpRules.length > 0) {
        return false;
      } else {
        return MESSAGES.noIpRules;
      }
    },
    { type: "confirm" },
  ),
  new ScenarioAction(
    "addInboundRule",
    /**
     * @param {{ defaultSecurityGroup: import('@aws-sdk/client-
ec2').SecurityGroup }} state
     */
    async (state) => {
      if (!state.shouldAddInboundRule) {
        return;
      }

      const client = new EC2Client({});
      await client.send(
        new AuthorizeSecurityGroupIngressCommand({
          GroupId: state.defaultSecurityGroup.GroupId,
          CidrIp: `${state.myIp}/32`,
          FromPort: 80,
          ToPort: 80,
          IpProtocol: "tcp",
        }),
      );
    },
  ),
```

```
  new ScenarioOutput("addedInboundRule", (state) => {
    if (state.shouldAddInboundRule) {
      return MESSAGES.addedInboundRule.replace("${IP_ADDRESS}", state.myIp);
    } else {
      return false;
    }
  }),
  new ScenarioOutput("verifyingEndpoint", (state) =>
    MESSAGES.verifyingEndpoint.replace("${DNS_NAME}", state.loadBalancerDns),
  ),
  new ScenarioAction("verifyEndpoint", async (state) => {
    try {
      const response = await retry({ intervalInMs: 2000, maxRetries: 30 }, () =>
        axios.get(`http://${state.loadBalancerDns}`),
      );
      state.endpointResponse = JSON.stringify(response.data, null, 2);
    } catch (e) {
      state.verifyEndpointError = e;
    }
  }),
  new ScenarioOutput("verifiedEndpoint", (state) => {
    if (state.verifyEndpointError) {
      console.error(state.verifyEndpointError);
    } else {
      return MESSAGES.verifiedEndpoint.replace(
        "${ENDPOINT_RESPONSE}",
        state.endpointResponse,
      );
    }
  }),
];
```

创建运行演示的步骤。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { readFileSync } from "node:fs";
import { join } from "node:path";

import axios from "axios";

import {
```

```
  DescribeTargetGroupsCommand,
  DescribeTargetHealthCommand,
  ElasticLoadBalancingV2Client,
} from "@aws-sdk/client-elastic-load-balancing-v2";
import {
  DescribeInstanceInformationCommand,
  PutParameterCommand,
  SSMClient,
  SendCommandCommand,
} from "@aws-sdk/client-ssm";
import {
  IAMClient,
  CreatePolicyCommand,
  CreateRoleCommand,
  AttachRolePolicyCommand,
  CreateInstanceProfileCommand,
  AddRoleToInstanceProfileCommand,
  waitUntilInstanceProfileExists,
} from "@aws-sdk/client-iam";
import {
  AutoScalingClient,
  DescribeAutoScalingGroupsCommand,
  TerminateInstanceInAutoScalingGroupCommand,
} from "@aws-sdk/client-auto-scaling";
import {
  DescribeIamInstanceProfileAssociationsCommand,
  EC2Client,
  RebootInstancesCommand,
  ReplaceIamInstanceProfileAssociationCommand,
} from "@aws-sdk/client-ec2";

import {
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/scenario.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES, RESOURCES_PATH } from "./constants.js";
import { findLoadBalancer } from "./shared.js";

const getRecommendation = new ScenarioAction(
  "getRecommendation",
  async (state) => {
```

```
    const loadBalancer = await findLoadBalancer(NAMES.loadBalancerName);
    if (loadBalancer) {
      state.loadBalancerDnsName = loadBalancer.DNSName;
      try {
        state.recommendation = (
          await axios.get(`http://${state.loadBalancerDnsName}`)
        ).data;
      } catch (e) {
        state.recommendation = e instanceof Error ? e.message : e;
      }
    } else {
      throw new Error(MESSAGES.demoFindLoadBalancerError);
    }
  },
);

const getRecommendationResult = new ScenarioOutput(
  "getRecommendationResult",
  (state) =>
    `Recommendation:\n${JSON.stringify(state.recommendation, null, 2)}`,
  { preformatted: true },
);

const getHealthCheck = new ScenarioAction("getHealthCheck", async (state) => {
  // snippet-start:[javascript.v3.wkflw.resilient.DescribeTargetGroups]
  const client = new ElasticLoadBalancingV2Client({});
  const { TargetGroups } = await client.send(
    new DescribeTargetGroupsCommand({
      Names: [NAMES.loadBalancerTargetGroupName],
    }),
  );
  // snippet-end:[javascript.v3.wkflw.resilient.DescribeTargetGroups]

  // snippet-start:[javascript.v3.wkflw.resilient.DescribeTargetHealth]
  const { TargetHealthDescriptions } = await client.send(
    new DescribeTargetHealthCommand({
      TargetGroupArn: TargetGroups[0].TargetGroupArn,
    }),
  );
  // snippet-end:[javascript.v3.wkflw.resilient.DescribeTargetHealth]
  state.targetHealthDescriptions = TargetHealthDescriptions;
});

const getHealthCheckResult = new ScenarioOutput(
```

```
  "getHealthCheckResult",
  /**
   * @param {{ targetHealthDescriptions: import('@aws-sdk/client-elastic-load-
balancing-v2').TargetHealthDescription[]}} state
   */
  (state) => {
    const status = state.targetHealthDescriptions
      .map((th) => `${th.Target.Id}: ${th.TargetHealth.State}`)
      .join("\n");
    return `Health check:\n${status}`;
  },
  { preformatted: true },
);

const loadBalancerLoop = new ScenarioAction(
  "loadBalancerLoop",
  getRecommendation.action,
  {
    whileConfig: {
      inputEquals: true,
      input: new ScenarioInput(
        "loadBalancerCheck",
        MESSAGES.demoLoadBalancerCheck,
        {
          type: "confirm",
        },
      ),
      output: getRecommendationResult,
    },
  },
);

const healthCheckLoop = new ScenarioAction(
  "healthCheckLoop",
  getHealthCheck.action,
  {
    whileConfig: {
      inputEquals: true,
      input: new ScenarioInput("healthCheck", MESSAGES.demoHealthCheck, {
        type: "confirm",
      }),
      output: getHealthCheckResult,
    },
  },
```

```
);

const statusSteps = [
  getRecommendation,
  getRecommendationResult,
  getHealthCheck,
  getHealthCheckResult,
];

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const demoSteps = [
  new ScenarioOutput("header", MESSAGES.demoHeader, { header: true }),
  new ScenarioOutput("sanityCheck", MESSAGES.demoSanityCheck),
  ...statusSteps,
  new ScenarioInput(
    "brokenDependencyConfirmation",
    MESSAGES.demoBrokenDependencyConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("brokenDependency", async (state) => {
    if (!state.brokenDependencyConfirmation) {
      process.exit();
    } else {
      const client = new SSMClient({});
      state.badTableName = `fake-table-${Date.now()}`;
      await client.send(
        new PutParameterCommand({
          Name: NAMES.ssmTableNameKey,
          Value: state.badTableName,
          Overwrite: true,
          Type: "String",
        }),
      );
    }
  }),
  new ScenarioOutput("testBrokenDependency", (state) =>
    MESSAGES.demoTestBrokenDependency.replace(
      "${TABLE_NAME}",
      state.badTableName,
    ),
  ),
  ...statusSteps,
```

```
  new ScenarioInput(
    "staticResponseConfirmation",
    MESSAGES.demoStaticResponseConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("staticResponse", async (state) => {
    if (!state.staticResponseConfirmation) {
      process.exit();
    } else {
      const client = new SSMClient({});
      await client.send(
        new PutParameterCommand({
          Name: NAMES.ssmFailureResponseKey,
          Value: "static",
          Overwrite: true,
          Type: "String",
        }),
      );
    }
  }),
  new ScenarioOutput("testStaticResponse", MESSAGES.demoTestStaticResponse),
  ...statusSteps,
  new ScenarioInput(
    "badCredentialsConfirmation",
    MESSAGES.demoBadCredentialsConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("badCredentialsExit", (state) => {
    if (!state.badCredentialsConfirmation) {
      process.exit();
    }
  }),
  new ScenarioAction("fixDynamoDBName", async () => {
    const client = new SSMClient({});
    await client.send(
      new PutParameterCommand({
        Name: NAMES.ssmTableNameKey,
        Value: NAMES.tableName,
        Overwrite: true,
        Type: "String",
      }),
    );
  }),
  new ScenarioAction(
```

```
    "badCredentials",
    /**
     * @param {{ targetInstance: import('@aws-sdk/client-auto-scaling').Instance }}
 state
     */
    async (state) => {
      await createSsmOnlyInstanceProfile();
      const autoScalingClient = new AutoScalingClient({});
      const { AutoScalingGroups } = await autoScalingClient.send(
        new DescribeAutoScalingGroupsCommand({
          AutoScalingGroupNames: [NAMES.autoScalingGroupName],
        }),
      );
      state.targetInstance = AutoScalingGroups[0].Instances[0];
      // snippet-start:
[javascript.v3.wkflw.resilient.DescribeIamInstanceProfileAssociations]
      const ec2Client = new EC2Client({});
      const { IamInstanceProfileAssociations } = await ec2Client.send(
        new DescribeIamInstanceProfileAssociationsCommand({
          Filters: [
            { Name: "instance-id", Values: [state.targetInstance.InstanceId] },
          ],
        }),
      );
      // snippet-end:
[javascript.v3.wkflw.resilient.DescribeIamInstanceProfileAssociations]
      state.instanceProfileAssociationId =
        IamInstanceProfileAssociations[0].AssociationId;
      // snippet-start:
[javascript.v3.wkflw.resilient.ReplaceIamInstanceProfileAssociation]
      await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
        ec2Client.send(
          new ReplaceIamInstanceProfileAssociationCommand({
            AssociationId: state.instanceProfileAssociationId,
            IamInstanceProfile: { Name: NAMES.ssmOnlyInstanceProfileName },
          }),
        ),
      );
      // snippet-end:
[javascript.v3.wkflw.resilient.ReplaceIamInstanceProfileAssociation]

      await ec2Client.send(
        new RebootInstancesCommand({
          InstanceIds: [state.targetInstance.InstanceId],
```

```
        }),
      );

      const ssmClient = new SSMClient({});
      await retry({ intervalInMs: 20000, maxRetries: 15 }, async () => {
        const { InstanceInformationList } = await ssmClient.send(
          new DescribeInstanceInformationCommand({}),
        );

        const instance = InstanceInformationList.find(
          (info) => info.InstanceId === state.targetInstance.InstanceId,
        );

        if (!instance) {
          throw new Error("Instance not found.");
        }
      });

      await ssmClient.send(
        new SendCommandCommand({
          InstanceIds: [state.targetInstance.InstanceId],
          DocumentName: "AWS-RunShellScript",
          Parameters: { commands: ["cd / && sudo python3 server.py 80"] },
        }),
      );
    },
  ),
  new ScenarioOutput(
    "testBadCredentials",
    /**
     * @param {{ targetInstance: import('@aws-sdk/client-ssm').InstanceInformation}}
 state
     */
    (state) =>
      MESSAGES.demoTestBadCredentials.replace(
        "${INSTANCE_ID}",
        state.targetInstance.InstanceId,
      ),
  ),
  loadBalancerLoop,
  new ScenarioInput(
    "deepHealthCheckConfirmation",
    MESSAGES.demoDeepHealthCheckConfirmation,
    { type: "confirm" },
```

```
  ),
  new ScenarioAction("deepHealthCheckExit", (state) => {
    if (!state.deepHealthCheckConfirmation) {
      process.exit();
    }
  }),
  new ScenarioAction("deepHealthCheck", async () => {
    const client = new SSMClient({});
    await client.send(
      new PutParameterCommand({
        Name: NAMES.ssmHealthCheckKey,
        Value: "deep",
        Overwrite: true,
        Type: "String",
      }),
    );
  }),
  new ScenarioOutput("testDeepHealthCheck", MESSAGES.demoTestDeepHealthCheck),
  healthCheckLoop,
  loadBalancerLoop,
  new ScenarioInput(
    "killInstanceConfirmation",
    /**
     * @param {{ targetInstance: import('@aws-sdk/client-
ssm').InstanceInformation }} state
     */
    (state) =>
      MESSAGES.demoKillInstanceConfirmation.replace(
        "${INSTANCE_ID}",
        state.targetInstance.InstanceId,
      ),
    { type: "confirm" },
  ),
  new ScenarioAction("killInstanceExit", (state) => {
    if (!state.killInstanceConfirmation) {
      process.exit();
    }
  }),
  new ScenarioAction(
    "killInstance",
    /**
     * @param {{ targetInstance: import('@aws-sdk/client-
ssm').InstanceInformation }} state
     */
```

```
    async (state) => {
      const client = new AutoScalingClient({});
      await client.send(
        new TerminateInstanceInAutoScalingGroupCommand({
          InstanceId: state.targetInstance.InstanceId,
          ShouldDecrementDesiredCapacity: false,
        }),
      );
    },
  ),
  new ScenarioOutput("testKillInstance", MESSAGES.demoTestKillInstance),
  healthCheckLoop,
  loadBalancerLoop,
  new ScenarioInput("failOpenConfirmation", MESSAGES.demoFailOpenConfirmation, {
    type: "confirm",
  }),
  new ScenarioAction("failOpenExit", (state) => {
    if (!state.failOpenConfirmation) {
      process.exit();
    }
  }),
  new ScenarioAction("failOpen", () => {
    const client = new SSMClient({});
    return client.send(
      new PutParameterCommand({
        Name: NAMES.ssmTableNameKey,
        Value: `fake-table-${Date.now()}`,
        Overwrite: true,
        Type: "String",
      }),
    );
  }),
  new ScenarioOutput("testFailOpen", MESSAGES.demoFailOpenTest),
  healthCheckLoop,
  loadBalancerLoop,
  new ScenarioInput(
    "resetTableConfirmation",
    MESSAGES.demoResetTableConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("resetTableExit", (state) => {
    if (!state.resetTableConfirmation) {
      process.exit();
    }
```

```
  }),
  new ScenarioAction("resetTable", async () => {
    const client = new SSMClient({});
    await client.send(
      new PutParameterCommand({
        Name: NAMES.ssmTableNameKey,
        Value: NAMES.tableName,
        Overwrite: true,
        Type: "String",
      }),
    );
  }),
  new ScenarioOutput("testResetTable", MESSAGES.demoTestResetTable),
  healthCheckLoop,
  loadBalancerLoop,
];

async function createSsmOnlyInstanceProfile() {
  const iamClient = new IAMClient({});
  const { Policy } = await iamClient.send(
    new CreatePolicyCommand({
      PolicyName: NAMES.ssmOnlyPolicyName,
      PolicyDocument: readFileSync(
        join(RESOURCES_PATH, "ssm_only_policy.json"),
      ),
    }),
  );
  await iamClient.send(
    new CreateRoleCommand({
      RoleName: NAMES.ssmOnlyRoleName,
      AssumeRolePolicyDocument: JSON.stringify({
        Version: "2012-10-17",
        Statement: [
          {
            Effect: "Allow",
            Principal: { Service: "ec2.amazonaws.com" },
            Action: "sts:AssumeRole",
          },
        ],
      }),
    }),
  );
  await iamClient.send(
    new AttachRolePolicyCommand({
```

```
      RoleName: NAMES.ssmOnlyRoleName,
      PolicyArn: Policy.Arn,
    }),
  );
  await iamClient.send(
    new AttachRolePolicyCommand({
      RoleName: NAMES.ssmOnlyRoleName,
      PolicyArn: "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore",
    }),
  );
  // snippet-start:[javascript.v3.wkflw.resilient.CreateInstanceProfile]
  const { InstanceProfile } = await iamClient.send(
    new CreateInstanceProfileCommand({
      InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
    }),
  );
  await waitUntilInstanceProfileExists(
    { client: iamClient },
    { InstanceProfileName: NAMES.ssmOnlyInstanceProfileName },
  );
  // snippet-end:[javascript.v3.wkflw.resilient.CreateInstanceProfile]
  await iamClient.send(
    new AddRoleToInstanceProfileCommand({
      InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
      RoleName: NAMES.ssmOnlyRoleName,
    }),
  );

  return InstanceProfile;
}
```

创建销毁所有资源的步骤。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { unlinkSync } from "node:fs";

import { DynamoDBClient, DeleteTableCommand } from "@aws-sdk/client-dynamodb";
import {
  EC2Client,
  DeleteKeyPairCommand,
  DeleteLaunchTemplateCommand,
```

```
} from "@aws-sdk/client-ec2";
import {
  IAMClient,
  DeleteInstanceProfileCommand,
  RemoveRoleFromInstanceProfileCommand,
  DeletePolicyCommand,
  DeleteRoleCommand,
  DetachRolePolicyCommand,
  paginateListPolicies,
} from "@aws-sdk/client-iam";
import {
  AutoScalingClient,
  DeleteAutoScalingGroupCommand,
  TerminateInstanceInAutoScalingGroupCommand,
  UpdateAutoScalingGroupCommand,
  paginateDescribeAutoScalingGroups,
} from "@aws-sdk/client-auto-scaling";
import {
  DeleteLoadBalancerCommand,
  DeleteTargetGroupCommand,
  DescribeTargetGroupsCommand,
  ElasticLoadBalancingV2Client,
} from "@aws-sdk/client-elastic-load-balancing-v2";

import {
  ScenarioOutput,
  ScenarioInput,
  ScenarioAction,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES } from "./constants.js";
import { findLoadBalancer } from "./shared.js";

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const destroySteps = [
  new ScenarioInput("destroy", MESSAGES.destroy, { type: "confirm" }),
  new ScenarioAction(
    "abort",
    (state) => state.destroy === false && process.exit(),
  ),
  new ScenarioAction("deleteTable", async (c) => {
```

```
    try {
      const client = new DynamoDBClient({});
      await client.send(new DeleteTableCommand({ TableName: NAMES.tableName }));
    } catch (e) {
      c.deleteTableError = e;
    }
  }),
  new ScenarioOutput("deleteTableResult", (state) => {
    if (state.deleteTableError) {
      console.error(state.deleteTableError);
      return MESSAGES.deleteTableError.replace(
        "${TABLE_NAME}",
        NAMES.tableName,
      );
    } else {
      return MESSAGES.deletedTable.replace("${TABLE_NAME}", NAMES.tableName);
    }
  }),
  new ScenarioAction("deleteKeyPair", async (state) => {
    try {
      const client = new EC2Client({});
      await client.send(
        new DeleteKeyPairCommand({ KeyName: NAMES.keyPairName }),
      );
      unlinkSync(`${NAMES.keyPairName}.pem`);
    } catch (e) {
      state.deleteKeyPairError = e;
    }
  }),
  new ScenarioOutput("deleteKeyPairResult", (state) => {
    if (state.deleteKeyPairError) {
      console.error(state.deleteKeyPairError);
      return MESSAGES.deleteKeyPairError.replace(
        "${KEY_PAIR_NAME}",
        NAMES.keyPairName,
      );
    } else {
      return MESSAGES.deletedKeyPair.replace(
        "${KEY_PAIR_NAME}",
        NAMES.keyPairName,
      );
    }
  }),
  new ScenarioAction("detachPolicyFromRole", async (state) => {
```

```
    try {
      const client = new IAMClient({});
      const policy = await findPolicy(NAMES.instancePolicyName);

      if (!policy) {
        state.detachPolicyFromRoleError = new Error(
          `Policy ${NAMES.instancePolicyName} not found.`,
        );
      } else {
        await client.send(
          new DetachRolePolicyCommand({
            RoleName: NAMES.instanceRoleName,
            PolicyArn: policy.Arn,
          }),
        );
      }
    } catch (e) {
      state.detachPolicyFromRoleError = e;
    }
  }),
  new ScenarioOutput("detachedPolicyFromRole", (state) => {
    if (state.detachPolicyFromRoleError) {
      console.error(state.detachPolicyFromRoleError);
      return MESSAGES.detachPolicyFromRoleError
        .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    } else {
      return MESSAGES.detachedPolicyFromRole
        .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    }
  }),
  new ScenarioAction("deleteInstancePolicy", async (state) => {
    const client = new IAMClient({});
    const policy = await findPolicy(NAMES.instancePolicyName);

    if (!policy) {
      state.deletePolicyError = new Error(
        `Policy ${NAMES.instancePolicyName} not found.`,
      );
    } else {
      return client.send(
        new DeletePolicyCommand({
          PolicyArn: policy.Arn,
```

```
        }),
      );
    }
  }),
  new ScenarioOutput("deletePolicyResult", (state) => {
    if (state.deletePolicyError) {
      console.error(state.deletePolicyError);
      return MESSAGES.deletePolicyError.replace(
        "${INSTANCE_POLICY_NAME}",
        NAMES.instancePolicyName,
      );
    } else {
      return MESSAGES.deletedPolicy.replace(
        "${INSTANCE_POLICY_NAME}",
        NAMES.instancePolicyName,
      );
    }
  }),
  new ScenarioAction("removeRoleFromInstanceProfile", async (state) => {
    try {
      const client = new IAMClient({});
      await client.send(
        new RemoveRoleFromInstanceProfileCommand({
          RoleName: NAMES.instanceRoleName,
          InstanceProfileName: NAMES.instanceProfileName,
        }),
      );
    } catch (e) {
      state.removeRoleFromInstanceProfileError = e;
    }
  }),
  new ScenarioOutput("removeRoleFromInstanceProfileResult", (state) => {
    if (state.removeRoleFromInstanceProfile) {
      console.error(state.removeRoleFromInstanceProfileError);
      return MESSAGES.removeRoleFromInstanceProfileError
        .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    } else {
      return MESSAGES.removedRoleFromInstanceProfile
        .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    }
  }),
  new ScenarioAction("deleteInstanceRole", async (state) => {
```

```
    try {
      const client = new IAMClient({});
      await client.send(
        new DeleteRoleCommand({
          RoleName: NAMES.instanceRoleName,
        }),
      );
    } catch (e) {
      state.deleteInstanceRoleError = e;
    }
  }),
  new ScenarioOutput("deleteInstanceRoleResult", (state) => {
    if (state.deleteInstanceRoleError) {
      console.error(state.deleteInstanceRoleError);
      return MESSAGES.deleteInstanceRoleError.replace(
        "${INSTANCE_ROLE_NAME}",
        NAMES.instanceRoleName,
      );
    } else {
      return MESSAGES.deletedInstanceRole.replace(
        "${INSTANCE_ROLE_NAME}",
        NAMES.instanceRoleName,
      );
    }
  }),
  new ScenarioAction("deleteInstanceProfile", async (state) => {
    try {
      // snippet-start:[javascript.v3.wkflw.resilient.DeleteInstanceProfile]
      const client = new IAMClient({});
      await client.send(
        new DeleteInstanceProfileCommand({
          InstanceProfileName: NAMES.instanceProfileName,
        }),
      );
      // snippet-end:[javascript.v3.wkflw.resilient.DeleteInstanceProfile]
    } catch (e) {
      state.deleteInstanceProfileError = e;
    }
  }),
  new ScenarioOutput("deleteInstanceProfileResult", (state) => {
    if (state.deleteInstanceProfileError) {
      console.error(state.deleteInstanceProfileError);
      return MESSAGES.deleteInstanceProfileError.replace(
        "${INSTANCE_PROFILE_NAME}",
```

```
        NAMES.instanceProfileName,
      );
    } else {
      return MESSAGES.deletedInstanceProfile.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.instanceProfileName,
      );
    }
  }),
  new ScenarioAction("deleteAutoScalingGroup", async (state) => {
    try {
      await terminateGroupInstances(NAMES.autoScalingGroupName);
      await retry({ intervalInMs: 60000, maxRetries: 60 }, async () => {
        await deleteAutoScalingGroup(NAMES.autoScalingGroupName);
      });
    } catch (e) {
      state.deleteAutoScalingGroupError = e;
    }
  }),
  new ScenarioOutput("deleteAutoScalingGroupResult", (state) => {
    if (state.deleteAutoScalingGroupError) {
      console.error(state.deleteAutoScalingGroupError);
      return MESSAGES.deleteAutoScalingGroupError.replace(
        "${AUTO_SCALING_GROUP_NAME}",
        NAMES.autoScalingGroupName,
      );
    } else {
      return MESSAGES.deletedAutoScalingGroup.replace(
        "${AUTO_SCALING_GROUP_NAME}",
        NAMES.autoScalingGroupName,
      );
    }
  }),
  new ScenarioAction("deleteLaunchTemplate", async (state) => {
    const client = new EC2Client({});
    try {
      // snippet-start:[javascript.v3.wkflw.resilient.DeleteLaunchTemplate]
      await client.send(
        new DeleteLaunchTemplateCommand({
          LaunchTemplateName: NAMES.launchTemplateName,
        }),
      );
      // snippet-end:[javascript.v3.wkflw.resilient.DeleteLaunchTemplate]
    } catch (e) {
```

```
      state.deleteLaunchTemplateError = e;
    }
  }),
  new ScenarioOutput("deleteLaunchTemplateResult", (state) => {
    if (state.deleteLaunchTemplateError) {
      console.error(state.deleteLaunchTemplateError);
      return MESSAGES.deleteLaunchTemplateError.replace(
        "${LAUNCH_TEMPLATE_NAME}",
        NAMES.launchTemplateName,
      );
    } else {
      return MESSAGES.deletedLaunchTemplate.replace(
        "${LAUNCH_TEMPLATE_NAME}",
        NAMES.launchTemplateName,
      );
    }
  }),
  new ScenarioAction("deleteLoadBalancer", async (state) => {
    try {
      // snippet-start:[javascript.v3.wkflw.resilient.DeleteLoadBalancer]
      const client = new ElasticLoadBalancingV2Client({});
      const loadBalancer = await findLoadBalancer(NAMES.loadBalancerName);
      await client.send(
        new DeleteLoadBalancerCommand({
          LoadBalancerArn: loadBalancer.LoadBalancerArn,
        }),
      );
      await retry({ intervalInMs: 1000, maxRetries: 60 }, async () => {
        const lb = await findLoadBalancer(NAMES.loadBalancerName);
        if (lb) {
          throw new Error("Load balancer still exists.");
        }
      });
      // snippet-end:[javascript.v3.wkflw.resilient.DeleteLoadBalancer]
    } catch (e) {
      state.deleteLoadBalancerError = e;
    }
  }),
  new ScenarioOutput("deleteLoadBalancerResult", (state) => {
    if (state.deleteLoadBalancerError) {
      console.error(state.deleteLoadBalancerError);
      return MESSAGES.deleteLoadBalancerError.replace(
        "${LB_NAME}",
        NAMES.loadBalancerName,
```

```
      );
    } else {
      return MESSAGES.deletedLoadBalancer.replace(
        "${LB_NAME}",
        NAMES.loadBalancerName,
      );
    }
  }),
  new ScenarioAction("deleteLoadBalancerTargetGroup", async (state) => {
    // snippet-start:[javascript.v3.wkflw.resilient.DeleteTargetGroup]
    const client = new ElasticLoadBalancingV2Client({});
    try {
      const { TargetGroups } = await client.send(
        new DescribeTargetGroupsCommand({
          Names: [NAMES.loadBalancerTargetGroupName],
        }),
      );

      await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
        client.send(
          new DeleteTargetGroupCommand({
            TargetGroupArn: TargetGroups[0].TargetGroupArn,
          }),
        ),
      );
    } catch (e) {
      state.deleteLoadBalancerTargetGroupError = e;
    }
    // snippet-end:[javascript.v3.wkflw.resilient.DeleteTargetGroup]
  }),
  new ScenarioOutput("deleteLoadBalancerTargetGroupResult", (state) => {
    if (state.deleteLoadBalancerTargetGroupError) {
      console.error(state.deleteLoadBalancerTargetGroupError);
      return MESSAGES.deleteLoadBalancerTargetGroupError.replace(
        "${TARGET_GROUP_NAME}",
        NAMES.loadBalancerTargetGroupName,
      );
    } else {
      return MESSAGES.deletedLoadBalancerTargetGroup.replace(
        "${TARGET_GROUP_NAME}",
        NAMES.loadBalancerTargetGroupName,
      );
    }
  }),
```

```
    new ScenarioAction("detachSsmOnlyRoleFromProfile", async (state) => {
      try {
        const client = new IAMClient({});
        await client.send(
          new RemoveRoleFromInstanceProfileCommand({
            InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
            RoleName: NAMES.ssmOnlyRoleName,
          }),
        );
      } catch (e) {
        state.detachSsmOnlyRoleFromProfileError = e;
      }
    }),
    new ScenarioOutput("detachSsmOnlyRoleFromProfileResult", (state) => {
      if (state.detachSsmOnlyRoleFromProfileError) {
        console.error(state.detachSsmOnlyRoleFromProfileError);
        return MESSAGES.detachSsmOnlyRoleFromProfileError
          .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
          .replace("${PROFILE_NAME}", NAMES.ssmOnlyInstanceProfileName);
      } else {
        return MESSAGES.detachedSsmOnlyRoleFromProfile
          .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
          .replace("${PROFILE_NAME}", NAMES.ssmOnlyInstanceProfileName);
      }
    }),
    new ScenarioAction("detachSsmOnlyCustomRolePolicy", async (state) => {
      try {
        const iamClient = new IAMClient({});
        const ssmOnlyPolicy = await findPolicy(NAMES.ssmOnlyPolicyName);
        await iamClient.send(
          new DetachRolePolicyCommand({
            RoleName: NAMES.ssmOnlyRoleName,
            PolicyArn: ssmOnlyPolicy.Arn,
          }),
        );
      } catch (e) {
        state.detachSsmOnlyCustomRolePolicyError = e;
      }
    }),
    new ScenarioOutput("detachSsmOnlyCustomRolePolicyResult", (state) => {
      if (state.detachSsmOnlyCustomRolePolicyError) {
        console.error(state.detachSsmOnlyCustomRolePolicyError);
        return MESSAGES.detachSsmOnlyCustomRolePolicyError
          .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
```

```
        .replace("${POLICY_NAME}", NAMES.ssmOnlyPolicyName);
    } else {
      return MESSAGES.detachedSsmOnlyCustomRolePolicy
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${POLICY_NAME}", NAMES.ssmOnlyPolicyName);
    }
  }),
  new ScenarioAction("detachSsmOnlyAWSRolePolicy", async (state) => {
    try {
      const iamClient = new IAMClient({});
      await iamClient.send(
        new DetachRolePolicyCommand({
          RoleName: NAMES.ssmOnlyRoleName,
          PolicyArn: "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore",
        }),
      );
    } catch (e) {
      state.detachSsmOnlyAWSRolePolicyError = e;
    }
  }),
  new ScenarioOutput("detachSsmOnlyAWSRolePolicyResult", (state) => {
    if (state.detachSsmOnlyAWSRolePolicyError) {
      console.error(state.detachSsmOnlyAWSRolePolicyError);
      return MESSAGES.detachSsmOnlyAWSRolePolicyError
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${POLICY_NAME}", "AmazonSSMManagedInstanceCore");
    } else {
      return MESSAGES.detachedSsmOnlyAWSRolePolicy
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${POLICY_NAME}", "AmazonSSMManagedInstanceCore");
    }
  }),
  new ScenarioAction("deleteSsmOnlyInstanceProfile", async (state) => {
    try {
      const iamClient = new IAMClient({});
      await iamClient.send(
        new DeleteInstanceProfileCommand({
          InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
        }),
      );
    } catch (e) {
      state.deleteSsmOnlyInstanceProfileError = e;
    }
  }),
```

```
  new ScenarioOutput("deleteSsmOnlyInstanceProfileResult", (state) => {
    if (state.deleteSsmOnlyInstanceProfileError) {
      console.error(state.deleteSsmOnlyInstanceProfileError);
      return MESSAGES.deleteSsmOnlyInstanceProfileError.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.ssmOnlyInstanceProfileName,
      );
    } else {
      return MESSAGES.deletedSsmOnlyInstanceProfile.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.ssmOnlyInstanceProfileName,
      );
    }
  }),
  new ScenarioAction("deleteSsmOnlyPolicy", async (state) => {
    try {
      const iamClient = new IAMClient({});
      const ssmOnlyPolicy = await findPolicy(NAMES.ssmOnlyPolicyName);
      await iamClient.send(
        new DeletePolicyCommand({
          PolicyArn: ssmOnlyPolicy.Arn,
        }),
      );
    } catch (e) {
      state.deleteSsmOnlyPolicyError = e;
    }
  }),
  new ScenarioOutput("deleteSsmOnlyPolicyResult", (state) => {
    if (state.deleteSsmOnlyPolicyError) {
      console.error(state.deleteSsmOnlyPolicyError);
      return MESSAGES.deleteSsmOnlyPolicyError.replace(
        "${POLICY_NAME}",
        NAMES.ssmOnlyPolicyName,
      );
    } else {
      return MESSAGES.deletedSsmOnlyPolicy.replace(
        "${POLICY_NAME}",
        NAMES.ssmOnlyPolicyName,
      );
    }
  }),
  new ScenarioAction("deleteSsmOnlyRole", async (state) => {
    try {
      const iamClient = new IAMClient({});
```

```
      await iamClient.send(
        new DeleteRoleCommand({
          RoleName: NAMES.ssmOnlyRoleName,
        }),
      );
    } catch (e) {
      state.deleteSsmOnlyRoleError = e;
    }
  }),
  new ScenarioOutput("deleteSsmOnlyRoleResult", (state) => {
    if (state.deleteSsmOnlyRoleError) {
      console.error(state.deleteSsmOnlyRoleError);
      return MESSAGES.deleteSsmOnlyRoleError.replace(
        "${ROLE_NAME}",
        NAMES.ssmOnlyRoleName,
      );
    } else {
      return MESSAGES.deletedSsmOnlyRole.replace(
        "${ROLE_NAME}",
        NAMES.ssmOnlyRoleName,
      );
    }
  }),
];

/**
 * @param {string} policyName
 */
async function findPolicy(policyName) {
  const client = new IAMClient({});
  const paginatedPolicies = paginateListPolicies({ client }, {});
  for await (const page of paginatedPolicies) {
    const policy = page.Policies.find((p) => p.PolicyName === policyName);
    if (policy) {
      return policy;
    }
  }
}

/**
 * @param {string} groupName
 */
async function deleteAutoScalingGroup(groupName) {
  const client = new AutoScalingClient({});
```

```
  try {
    await client.send(
      new DeleteAutoScalingGroupCommand({
        AutoScalingGroupName: groupName,
      }),
    );
  } catch (err) {
    if (!(err instanceof Error)) {
      throw err;
    } else {
      console.log(err.name);
      throw err;
    }
  }
}

/**
 * @param {string} groupName
 */
async function terminateGroupInstances(groupName) {
  const autoScalingClient = new AutoScalingClient({});
  const group = await findAutoScalingGroup(groupName);
  await autoScalingClient.send(
    new UpdateAutoScalingGroupCommand({
      AutoScalingGroupName: group.AutoScalingGroupName,
      MinSize: 0,
    }),
  );
  for (const i of group.Instances) {
    await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
      autoScalingClient.send(
        new TerminateInstanceInAutoScalingGroupCommand({
          InstanceId: i.InstanceId,
          ShouldDecrementDesiredCapacity: true,
        }),
      ),
    );
  }
}

async function findAutoScalingGroup(groupName) {
  const client = new AutoScalingClient({});
  const paginatedGroups = paginateDescribeAutoScalingGroups({ client }, {});
  for await (const page of paginatedGroups) {
```

```
    const group = page.AutoScalingGroups.find(
      (g) => g.AutoScalingGroupName === groupName,
    );
    if (group) {
      return group;
    }
  }
  throw new Error(`Auto scaling group ${groupName} not found.`);
}
```

- 有关 API 详细信息，请参阅《Amazon SDK for JavaScript API 参考》中的以下主题。
  - AttachLoadBalancerTargetGroups
  - CreateAutoScalingGroup
  - CreateInstanceProfile
  - CreateLaunchTemplate
  - CreateListener
  - CreateLoadBalancer
  - CreateTargetGroup
  - DeleteAutoScalingGroup
  - DeleteInstanceProfile
  - DeleteLaunchTemplate
  - DeleteLoadBalancer
  - DeleteTargetGroup
  - DescribeAutoScalingGroups
  - DescribeAvailabilityZones
  - DescribeIamInstanceProfileAssociations
  - DescribeInstances
  - DescribeLoadBalancers
  - DescribeSubnets
  - DescribeTargetGroups
  - DescribeTargetHealth
  - DescribeVpcs
  - RebootInstances

- ReplaceIamInstanceProfileAssociation
- TerminateInstanceInAutoScalingGroup
- UpdateAutoScalingGroup

创建用户并代入角色

以下代码示例展示了如何创建用户并代入角色。

Warning

为了避免安全风险，在开发专用软件或处理真实数据时，请勿使用 IAM 用户进行身份验证，而是使用与身份提供商的联合身份验证，例如 Amazon IAM Identity Center。

- 创建没有权限的用户。
- 创建授予列出账户的 Amazon S3 存储桶的权限的角色
- 添加策略以允许用户代入该角色。
- 代入角色并使用临时凭证列出 S3 存储桶，然后清除资源。

适用于 JavaScript (v3) 的软件开发工具包

Note

还有更多相关信息 GitHub。查找完整示例，学习如何在 Amazon 代码示例存储库中进行设置和运行。

创建 IAM 用户和授予列出 Amazon S3 存储桶的权限的角色。用户仅具有代入该角色的权限。代入该角色后，使用临时凭证列出该账户的存储桶。

```
import {
  CreateUserCommand,
  CreateAccessKeyCommand,
  CreatePolicyCommand,
  CreateRoleCommand,
  AttachRolePolicyCommand,
  DeleteAccessKeyCommand,
  DeleteUserCommand,
```

```
  DeleteRoleCommand,
  DeletePolicyCommand,
  DetachRolePolicyCommand,
  IAMClient,
} from "@aws-sdk/client-iam";
import { ListBucketsCommand, S3Client } from "@aws-sdk/client-s3";
import { AssumeRoleCommand, STSClient } from "@aws-sdk/client-sts";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

// Set the parameters.
const iamClient = new IAMClient({});
const userName = "test_name";
const policyName = "test_policy";
const roleName = "test_role";

export const main = async () => {
  // Create a user. The user has no permissions by default.
  const { User } = await iamClient.send(
    new CreateUserCommand({ UserName: userName }),
  );

  if (!User) {
    throw new Error("User not created");
  }

  // Create an access key. This key is used to authenticate the new user to
  // Amazon Simple Storage Service (Amazon S3) and AWS Security Token Service (AWS
 STS).
  // It's not best practice to use access keys. For more information, see https://
aws.amazon.com/iam/resources/best-practices/.
  const createAccessKeyResponse = await iamClient.send(
    new CreateAccessKeyCommand({ UserName: userName }),
  );

  if (
    !createAccessKeyResponse.AccessKey?.AccessKeyId ||
    !createAccessKeyResponse.AccessKey?.SecretAccessKey
  ) {
    throw new Error("Access key not created");
  }

  const {
    AccessKey: { AccessKeyId, SecretAccessKey },
  } = createAccessKeyResponse;
```

```
  let s3Client = new S3Client({
    credentials: {
      accessKeyId: AccessKeyId,
      secretAccessKey: SecretAccessKey,
    },
  });

  // Retry the list buckets operation until it succeeds. InvalidAccessKeyId is
  // thrown while the user and access keys are still stabilizing.
  await retry({ intervalInMs: 1000, maxRetries: 300 }, async () => {
    try {
      return await listBuckets(s3Client);
    } catch (err) {
      if (err instanceof Error && err.name === "InvalidAccessKeyId") {
        throw err;
      }
    }
  });

  // Retry the create role operation until it succeeds. A MalformedPolicyDocument
 error
  // is thrown while the user and access keys are still stabilizing.
  const { Role } = await retry(
    {
      intervalInMs: 2000,
      maxRetries: 60,
    },
    () =>
      iamClient.send(
        new CreateRoleCommand({
          AssumeRolePolicyDocument: JSON.stringify({
            Version: "2012-10-17",
            Statement: [
              {
                Effect: "Allow",
                Principal: {
                  // Allow the previously created user to assume this role.
                  AWS: User.Arn,
                },
                Action: "sts:AssumeRole",
              },
            ],
          }),
```

```
          RoleName: roleName,
        }),
      ),
    );

    if (!Role) {
      throw new Error("Role not created");
    }

    // Create a policy that allows the user to list S3 buckets.
    const { Policy: listBucketPolicy } = await iamClient.send(
      new CreatePolicyCommand({
        PolicyDocument: JSON.stringify({
          Version: "2012-10-17",
          Statement: [
            {
              Effect: "Allow",
              Action: ["s3:ListAllMyBuckets"],
              Resource: "*",
            },
          ],
        }),
        PolicyName: policyName,
      }),
    );

    if (!listBucketPolicy) {
      throw new Error("Policy not created");
    }

    // Attach the policy granting the 's3:ListAllMyBuckets' action to the role.
    await iamClient.send(
      new AttachRolePolicyCommand({
        PolicyArn: listBucketPolicy.Arn,
        RoleName: Role.RoleName,
      }),
    );

    // Assume the role.
    const stsClient = new STSClient({
      credentials: {
        accessKeyId: AccessKeyId,
        secretAccessKey: SecretAccessKey,
      },
```

```
  });

  // Retry the assume role operation until it succeeds.
  const { Credentials } = await retry(
    { intervalInMs: 2000, maxRetries: 60 },
    () =>
      stsClient.send(
        new AssumeRoleCommand({
          RoleArn: Role.Arn,
          RoleSessionName: `iamBasicScenarioSession-${Math.floor(
            Math.random() * 1000000,
          )}`,
          DurationSeconds: 900,
        }),
      ),
  );

  if (!Credentials?.AccessKeyId || !Credentials?.SecretAccessKey) {
    throw new Error("Credentials not created");
  }

  s3Client = new S3Client({
    credentials: {
      accessKeyId: Credentials.AccessKeyId,
      secretAccessKey: Credentials.SecretAccessKey,
      sessionToken: Credentials.SessionToken,
    },
  });

  // List the S3 buckets again.
  // Retry the list buckets operation until it succeeds. AccessDenied might
  // be thrown while the role policy is still stabilizing.
  await retry({ intervalInMs: 2000, maxRetries: 60 }, () =>
    listBuckets(s3Client),
  );

  // Clean up.
  await iamClient.send(
    new DetachRolePolicyCommand({
      PolicyArn: listBucketPolicy.Arn,
      RoleName: Role.RoleName,
    }),
  );
```

```
  await iamClient.send(
    new DeletePolicyCommand({
      PolicyArn: listBucketPolicy.Arn,
    }),
  );

  await iamClient.send(
    new DeleteRoleCommand({
      RoleName: Role.RoleName,
    }),
  );

  await iamClient.send(
    new DeleteAccessKeyCommand({
      UserName: userName,
      AccessKeyId,
    }),
  );

  await iamClient.send(
    new DeleteUserCommand({
      UserName: userName,
    }),
  );
};

/**
 *
 * @param {S3Client} s3Client
 */
const listBuckets = async (s3Client) => {
  const { Buckets } = await s3Client.send(new ListBucketsCommand({}));

  if (!Buckets) {
    throw new Error("Buckets not listed");
  }

  console.log(Buckets.map((bucket) => bucket.Name).join("\n"));
};
```

- 有关 API 详细信息，请参阅《Amazon SDK for JavaScript API 参考》中的以下主题。

- AttachRolePolicy
- CreateAccessKey
- CreatePolicy
- CreateRole
- CreateUser
- DeleteAccessKey
- DeletePolicy
- DeleteRole
- DeleteUser
- DeleteUserPolicy
- DetachRolePolicy
- PutUserPolicy

使用适用于 JavaScript (v3) 的软件开发工具包的 Lambda 示例

以下代码示例向您展示了如何使用带有 Lambda 的 Amazon SDK for JavaScript (v3) 来执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景 是展示如何通过在同一服务中调用多个函数来完成特定任务的代码示例。

每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关如何在上下文中设置和运行代码的说明。

开始使用

开始使用 Lambda

以下代码示例展示了如何开始使用 Lambda。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { LambdaClient, paginateListFunctions } from "@aws-sdk/client-lambda";

const client = new LambdaClient({});

export const helloLambda = async () => {
  const paginator = paginateListFunctions({ client }, {});
  const functions = [];

  for await (const page of paginator) {
    const funcNames = page.Functions.map((f) => f.FunctionName);
    functions.push(...funcNames);
  }

  console.log("Functions:");
  console.log(functions.join("\n"));
  return functions;
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考ListFunctions中的。

主题

- 操作
- 场景

操作

创建函数

以下代码示例展示了如何创建 Lambda 函数。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
const createFunction = async (funcName, roleArn) => {
  const client = new LambdaClient({});
  const code = await readFile(`${dirname}../functions/${funcName}.zip`);

  const command = new CreateFunctionCommand({
    Code: { ZipFile: code },
    FunctionName: funcName,
    Role: roleArn,
    Architectures: [Architecture.arm64],
    Handler: "index.handler", // Required when sending a .zip file
    PackageType: PackageType.Zip, // Required when sending a .zip file
    Runtime: Runtime.nodejs16x, // Required when sending a .zip file
  });

  return client.send(command);
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考CreateFunction中的。

删除函数

以下代码示例展示了如何删除 Lambda 函数。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
/**
 * @param {string} funcName
 */
const deleteFunction = (funcName) => {
  const client = new LambdaClient({});
  const command = new DeleteFunctionCommand({ FunctionName: funcName });
  return client.send(command);
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DeleteFunction中的。

获取函数

以下代码示例展示了如何获取 Lambda 函数。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
const getFunction = (funcName) => {
  const client = new LambdaClient({});
  const command = new GetFunctionCommand({ FunctionName: funcName });
  return client.send(command);
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考GetFunction中的。

调用函数

以下代码示例展示了如何调用 Lambda 函数。

适用于 JavaScript (v3) 的软件开发工具包

Note

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
const invoke = async (funcName, payload) => {
  const client = new LambdaClient({});
  const command = new InvokeCommand({
    FunctionName: funcName,
    Payload: JSON.stringify(payload),
    LogType: LogType.Tail,
  });

  const { Payload, LogResult } = await client.send(command);
  const result = Buffer.from(Payload).toString();
  const logs = Buffer.from(LogResult, "base64").toString();
  return { logs, result };
};
```

- 有关 API 详细信息，请参阅《Amazon SDK for JavaScript API 参考》中的 Invoke。

列出函数

以下代码示例展示了如何列出 Lambda 函数。

适用于 JavaScript (v3) 的软件开发工具包

Note

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
const listFunctions = () => {
  const client = new LambdaClient({});
  const command = new ListFunctionsCommand({});
```

```
  return client.send(command);
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考ListFunctions中的。

更新函数代码

以下代码示例展示了如何更新 Lambda 函数代码。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
const updateFunctionCode = async (funcName, newFunc) => {
  const client = new LambdaClient({});
  const code = await readFile(`${dirname}../functions/${newFunc}.zip`);
  const command = new UpdateFunctionCodeCommand({
    ZipFile: code,
    FunctionName: funcName,
    Architectures: [Architecture.arm64],
    Handler: "index.handler", // Required when sending a .zip file
    PackageType: PackageType.Zip, // Required when sending a .zip file
    Runtime: Runtime.nodejs16x, // Required when sending a .zip file
  });

  return client.send(command);
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考UpdateFunctionCode中的。

更新函数配置

以下代码示例展示了如何更新 Lambda 函数配置。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
const updateFunctionConfiguration = (funcName) => {
  const client = new LambdaClient({});
  const config = readFileSync(`${dirname}../functions/config.json`).toString();
  const command = new UpdateFunctionConfigurationCommand({
    ...JSON.parse(config),
    FunctionName: funcName,
  });
  return client.send(command);
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考UpdateFunctionConfiguration中的。

场景

函数入门

以下代码示例展示了如何：

- 创建 IAM 角色和 Lambda 函数，然后上传处理程序代码。
- 使用单个参数来调用函数并获取结果。
- 更新函数代码并使用环境变量进行配置。
- 使用新参数来调用函数并获取结果。显示返回的执行日志。
- 列出账户函数，然后清除函数。

有关更多信息，请参阅使用控制台创建 Lambda 函数。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

创建一个 Amazon Identity and Access Management (IAM) 角色以授予 Lambda 写入日志的权限。

```
    log(`Creating role (${NAME_ROLE_LAMBDA})...`);
    const response = await createRole(NAME_ROLE_LAMBDA);

import { AttachRolePolicyCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} policyArn
 * @param {string} roleName
 */
export const attachRolePolicy = (policyArn, roleName) => {
  const command = new AttachRolePolicyCommand({
    PolicyArn: policyArn,
    RoleName: roleName,
  });

  return client.send(command);
};
```

创建 Lambda 函数并上传处理程序代码。

```
const createFunction = async (funcName, roleArn) => {
  const client = new LambdaClient({});
  const code = await readFile(`${dirname}../functions/${funcName}.zip`);

  const command = new CreateFunctionCommand({
    Code: { ZipFile: code },
    FunctionName: funcName,
    Role: roleArn,
```

```
    Architectures: [Architecture.arm64],
    Handler: "index.handler", // Required when sending a .zip file
    PackageType: PackageType.Zip, // Required when sending a .zip file
    Runtime: Runtime.nodejs16x, // Required when sending a .zip file
  });

  return client.send(command);
};
```

调用单参数函数并得出结果。

```
const invoke = async (funcName, payload) => {
  const client = new LambdaClient({});
  const command = new InvokeCommand({
    FunctionName: funcName,
    Payload: JSON.stringify(payload),
    LogType: LogType.Tail,
  });

  const { Payload, LogResult } = await client.send(command);
  const result = Buffer.from(Payload).toString();
  const logs = Buffer.from(LogResult, "base64").toString();
  return { logs, result };
};
```

更新函数代码并使用环境变量配置其 Lambda 环境。

```
const updateFunctionCode = async (funcName, newFunc) => {
  const client = new LambdaClient({});
  const code = await readFile(`${dirname}../functions/${newFunc}.zip`);
  const command = new UpdateFunctionCodeCommand({
    ZipFile: code,
    FunctionName: funcName,
    Architectures: [Architecture.arm64],
    Handler: "index.handler", // Required when sending a .zip file
    PackageType: PackageType.Zip, // Required when sending a .zip file
    Runtime: Runtime.nodejs16x, // Required when sending a .zip file
  });

  return client.send(command);
};
```

```
const updateFunctionConfiguration = (funcName) => {
  const client = new LambdaClient({});
  const config = readFileSync(`${dirname}../functions/config.json`).toString();
  const command = new UpdateFunctionConfigurationCommand({
    ...JSON.parse(config),
    FunctionName: funcName,
  });
  return client.send(command);
};
```

列出您账户的函数。

```
const listFunctions = () => {
  const client = new LambdaClient({});
  const command = new ListFunctionsCommand({});

  return client.send(command);
};
```

删除 IAM 角色和 Lambda 函数。

```
import { DeleteRoleCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} roleName
 */
export const deleteRole = (roleName) => {
  const command = new DeleteRoleCommand({ RoleName: roleName });
  return client.send(command);
};

/**
 * @param {string} funcName
 */
const deleteFunction = (funcName) => {
  const client = new LambdaClient({});
  const command = new DeleteFunctionCommand({ FunctionName: funcName });
```

```
  return client.send(command);
};
```

- 有关 API 详细信息，请参阅《Amazon SDK for JavaScript API 参考》中的以下主题。
  - CreateFunction
  - DeleteFunction
  - GetFunction
  - Invoke
  - ListFunctions
  - UpdateFunctionCode
  - UpdateFunctionConfiguration

Amazon 使用适用于 JavaScript (v3) 的软件开发工具包对示例进行个性化设置

以下代码示例向您展示了如何使用带有 Amazon Personalize 的 Amazon SDK for JavaScript (v3) 来执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景 是展示如何通过在同一服务中调用多个函数来完成特定任务的代码示例。

每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关如何在上下文中设置和运行代码的说明。

主题

- 操作

操作

创建批量接口作业

以下代码示例展示了如何创建 Amazon Personalize 批量接口作业。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
// Get service clients module and commands using ES6 syntax.
import { CreateBatchInferenceJobCommand } from
  "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the batch inference job's parameters.

export const createBatchInferenceJobParam = {
  jobName: 'JOB_NAME',
  jobInput: {         /* required */
    s3DataSource: {   /* required */
      path: 'INPUT_PATH', /* required */
      // kmsKeyArn: 'INPUT_KMS_KEY_ARN' /* optional */'
    }
  },
  jobOutput: {         /* required */
    s3DataDestination: {   /* required */
      path: 'OUTPUT_PATH', /* required */
      // kmsKeyArn: 'OUTPUT_KMS_KEY_ARN' /* optional */'
    }
  },
  roleArn: 'ROLE_ARN', /* required */
  solutionVersionArn: 'SOLUTION_VERSION_ARN', /* required */
  numResults: 20 /* optional integer*/
};

export const run = async () => {
  try {
    const response = await personalizeClient.send(new
 CreateBatchInferenceJobCommand(createBatchInferenceJobParam));
    console.log("Success", response);
```

```
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考CreateBatchInferenceJob中的。

创建批量分段作业

以下代码示例展示了如何创建 Amazon Personalize 批量分段作业。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
// Get service clients module and commands using ES6 syntax.
import { CreateBatchSegmentJobCommand } from
  "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the batch segment job's parameters.

export const createBatchSegmentJobParam = {
  jobName: 'NAME',
  jobInput: {          /* required */
    s3DataSource: {    /* required */
      path: 'INPUT_PATH', /* required */
      // kmsKeyArn: 'INPUT_KMS_KEY_ARN' /* optional */'
    }
  },
```

```
  jobOutput: {          /* required */
    s3DataDestination: {   /* required */
      path: 'OUTPUT_PATH', /* required */
      // kmsKeyArn: 'OUTPUT_KMS_KEY_ARN' /* optional */'
    }
  },
  roleArn: 'ROLE_ARN', /* required */
  solutionVersionArn: 'SOLUTION_VERSION_ARN', /* required */
  numResults: 20 /* optional */
};

export const run = async () => {
  try {
    const response = await personalizeClient.send(new
 CreateBatchSegmentJobCommand(createBatchSegmentJobParam));
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考CreateBatchSegmentJob中的。

创建活动

以下代码示例展示了如何创建 Amazon Personalize 市场活动。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
// Get service clients module and commands using ES6 syntax.

import { CreateCampaignCommand } from
```

```
  "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the campaign's parameters.
export const createCampaignParam = {
  solutionVersionArn: 'SOLUTION_VERSION_ARN', /* required */
  name: 'NAME',  /* required */
  minProvisionedTPS: 1    /* optional integer */
}

export const run = async () => {
  try {
    const response = await personalizeClient.send(new
 CreateCampaignCommand(createCampaignParam));
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考CreateCampaign中的。

创建数据集

以下代码示例展示了如何创建 Amazon Personalize 数据集。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
// Get service clients module and commands using ES6 syntax.
import { CreateDatasetCommand } from
```

```
  "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the dataset's parameters.
export const createDatasetParam = {
  datasetGroupArn: 'DATASET_GROUP_ARN', /* required */
  datasetType: 'DATASET_TYPE', /* required */
  name: 'NAME', /* required */
  schemaArn: 'SCHEMA_ARN' /* required */
}

export const run = async () => {
  try {
    const response = await personalizeClient.send(new
 CreateDatasetCommand(createDatasetParam));
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考CreateDataset中的。

创建数据集导出作业

以下代码示例展示了如何创建 Amazon Personalize 数据集导出作业。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
// Get service clients module and commands using ES6 syntax.
```

```
import { CreateDatasetExportJobCommand } from
  "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the export job parameters.
export const datasetExportJobParam = {
  datasetArn: 'DATASET_ARN', /* required */
  jobOutput: {
    s3DataDestination: {
        path: 'S3_DESTINATION_PATH' /* required */
        //kmsKeyArn: 'ARN'  /* include if your bucket uses AWS KMS for encryption
    }
  },
  jobName: 'NAME',/* required */
  roleArn: 'ROLE_ARN' /* required */
}

export const run = async () => {
  try {
    const response = await personalizeClient.send(new
 CreateDatasetExportJobCommand(datasetExportJobParam));
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考[CreateDatasetExportJob](CreateDatasetExportJob)中的。

创建数据集组

以下代码示例展示了如何创建 Amazon Personalize 数据集组。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
// Get service clients module and commands using ES6 syntax.

import { CreateDatasetGroupCommand } from
  "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the dataset group parameters.
export const createDatasetGroupParam = {
  name: 'NAME' /* required */
}

export const run = async (createDatasetGroupParam) => {
  try {
    const response = await personalizeClient.send(new
 CreateDatasetGroupCommand(createDatasetGroupParam));
    console.log("Success", response);
    return "Run successfully"; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run(createDatasetGroupParam);
```

创建域数据集组。

```
// Get service clients module and commands using ES6 syntax.
import {  CreateDatasetGroupCommand } from
  "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";
```

```
// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the domain dataset group parameters.
export const domainDatasetGroupParams = {
  name: 'NAME',  /* required */
  domain: 'DOMAIN'   /* required for a domain dsg, specify ECOMMERCE or
 VIDEO_ON_DEMAND */
}

export const run = async () => {
  try {
    const response = await personalizeClient.send(new
 CreateDatasetGroupCommand(domainDatasetGroupParams));
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考CreateDatasetGroup中的。

创建数据集导入作业

以下代码示例展示了如何创建 Amazon Personalize 数据集导入作业。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
// Get service clients module and commands using ES6 syntax.
import {CreateDatasetImportJobCommand } from
  "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";
```

```
// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the dataset import job parameters.
export const datasetImportJobParam = {
  datasetArn: 'DATASET_ARN', /* required */
  dataSource: {  /* required */
    dataLocation: 'S3_PATH'
  },
  jobName: 'NAME',/* required */
  roleArn: 'ROLE_ARN' /* required */
}

export const run = async () => {
  try {
    const response = await personalizeClient.send(new
 CreateDatasetImportJobCommand(datasetImportJobParam));
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考CreateDatasetImportJob中的。

创建域架构

以下代码示例展示了如何创建 Amazon Personalize 域架构。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
// Get service clients module and commands using ES6 syntax.
```

```
import { CreateSchemaCommand } from
  "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

import fs from 'fs';

let schemaFilePath = "SCHEMA_PATH";
let mySchema = "";

try {
  mySchema = fs.readFileSync(schemaFilePath).toString();
} catch (err) {
  mySchema = 'TEST' // for unit tests.
}

// Set the domain schema parameters.
export const createDomainSchemaParam = {
  name: 'NAME', /* required */
  schema: mySchema, /* required */
  domain: 'DOMAIN'   /* required for a domain dataset group, specify ECOMMERCE or
 VIDEO_ON_DEMAND */
};

export const run = async () => {
  try {
    const response = await personalizeClient.send(new
 CreateSchemaCommand(createDomainSchemaParam));
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考CreateSchema中的。

创建筛选条件

以下代码示例展示了如何创建 Amazon Personalize 筛选条件。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
// Get service clients module and commands using ES6 syntax.
import { CreateFilterCommand } from
  "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";
// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the filter's parameters.
export const createFilterParam = {
  datasetGroupArn: 'DATASET_GROUP_ARN', /* required */
  name: 'NAME', /* required */
  filterExpression: 'FILTER_EXPRESSION' /*required */
}

export const run = async () => {
  try {
    const response = await personalizeClient.send(new
 CreateFilterCommand(createFilterParam));
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考CreateFilter中的。

创建推荐系统

以下代码示例展示了如何创建 Amazon Personalize 推荐系统。

适用于 JavaScript (v3) 的软件开发工具包

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
// Get service clients module and commands using ES6 syntax.
import { CreateRecommenderCommand } from
  "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the recommender's parameters.
export const createRecommenderParam = {
  name: 'NAME', /* required */
  recipeArn: 'RECIPE_ARN', /* required */
  datasetGroupArn: 'DATASET_GROUP_ARN'  /* required */
}

export const run = async () => {
  try {
    const response = await personalizeClient.send(new
 CreateRecommenderCommand(createRecommenderParam));
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考CreateRecommender中的。

创建架构

以下代码示例展示了如何创建 Amazon Personalize 架构。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
// Get service clients module and commands using ES6 syntax.
import { CreateSchemaCommand } from
  "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

import fs from 'fs';

let schemaFilePath = "SCHEMA_PATH";
let mySchema = "";

try {
  mySchema = fs.readFileSync(schemaFilePath).toString();
} catch (err) {
  mySchema = 'TEST' // For unit tests.
}
// Set the schema parameters.
export const createSchemaParam = {
  name: 'NAME', /* required */
  schema: mySchema /* required */
};

export const run = async () => {
  try {
    const response = await personalizeClient.send(new
 CreateSchemaCommand(createSchemaParam));
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
```

```
    console.log("Error", err);
  }
};
run();
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考CreateSchema中的。

创建解决方案

以下代码示例展示了如何创建 Amazon Personalize 解决方案。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
// Get service clients module and commands using ES6 syntax.
import { CreateSolutionCommand } from
  "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";
// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the solution parameters.
export const createSolutionParam = {
  datasetGroupArn: 'DATASET_GROUP_ARN', /* required */
  recipeArn: 'RECIPE_ARN', /* required */
  name: 'NAME' /* required */
}

export const run = async () => {
  try {
    const response = await personalizeClient.send(new
 CreateSolutionCommand(createSolutionParam));
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
```

```
  }
};
run();
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考CreateSolution中的。

创建解决方案版本

以下代码示例展示了如何创建 Amazon Personalize 解决方案。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
// Get service clients module and commands using ES6 syntax.
import { CreateSolutionVersionCommand } from
  "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";
// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the solution version parameters.
export const solutionVersionParam = {
  solutionArn: 'SOLUTION_ARN' /* required */
}

export const run = async () => {
  try {
    const response = await personalizeClient.send(new
 CreateSolutionVersionCommand(solutionVersionParam));
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考CreateSolutionVersion中的。

创建事件跟踪器

以下代码示例展示了如何创建 Amazon Personalize 事件跟踪器。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
// Get service clients module and commands using ES6 syntax.
import { CreateEventTrackerCommand } from
  "@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the event tracker's parameters.
export const createEventTrackerParam = {
  datasetGroupArn: 'DATASET_GROUP_ARN', /* required */
  name: 'NAME', /* required */
}

export const run = async () => {
  try {
    const response = await personalizeClient.send(new
 CreateEventTrackerCommand(createEventTrackerParam));
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考CreateEventTracker中的。

Amazon 使用适用于 JavaScript (v3) 的软件开发工具包对事件进行个性化设置示例

以下代码示例向您展示了如何使用带有 Amazon Personalize Events 的 Amazon SDK for JavaScript (v3) 来执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景 是展示如何通过在同一服务中调用多个函数来完成特定任务的代码示例。

每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关如何在上下文中设置和运行代码的说明。

主题

- 操作

操作

将项目导入数据集

以下代码示例展示了如何以增量方式将项目导入 Amazon Personalize Events 数据集。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
// Get service clients module and commands using ES6 syntax.
import { PutItemsCommand } from "@aws-sdk/client-personalize-events";
import { personalizeEventsClient } from "./libs/personalizeClients.js";
// Or, create the client here.
```

```
// const personalizeEventsClient = new PersonalizeEventsClient({ region: "REGION"});

// Set the put items parameters. For string properties and values, use the \
 character to escape quotes.
var putItemsParam = {
  datasetArn: "DATASET_ARN" /* required */,
  items: [
    /* required */
    {
      itemId: "ITEM_ID" /*  required */,
      properties:
        '{"PROPERTY1_NAME": "PROPERTY1_VALUE", "PROPERTY2_NAME": "PROPERTY2_VALUE",
 "PROPERTY3_NAME": "PROPERTY3_VALUE"}' /* optional */,
    },
  ],
};
export const run = async () => {
  try {
    const response = await personalizeEventsClient.send(
      new PutItemsCommand(putItemsParam),
    );
    console.log("Success!", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考PutItems中的。

导入实时互动事件数据

以下代码示例展示了如何将实时互动事件数据导入 Amazon Personalize Events。

适用于 JavaScript (v3) 的软件开发工具包

Note

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
// Get service clients module and commands using ES6 syntax.
import { PutEventsCommand } from "@aws-sdk/client-personalize-events";
import { personalizeEventsClient } from "./libs/personalizeClients.js";
// Or, create the client here.
// const personalizeEventsClient = new PersonalizeEventsClient({ region: "REGION"});

// Convert your UNIX timestamp to a Date.
const sentAtDate = new Date(1613443801 * 1000); // 1613443801 is a testing value.
 Replace it with your sentAt timestamp in UNIX format.

// Set put events parameters.
var putEventsParam = {
  eventList: [
    /* required */
    {
      eventType: "EVENT_TYPE" /* required */,
      sentAt: sentAtDate /* required, must be a Date with js */,
      eventId: "EVENT_ID" /* optional */,
      itemId: "ITEM_ID" /* optional */,
    },
  ],
  sessionId: "SESSION_ID" /* required */,
  trackingId: "TRACKING_ID" /* required */,
  userId: "USER_ID" /* required */,
};
export const run = async () => {
  try {
    const response = await personalizeEventsClient.send(
      new PutEventsCommand(putEventsParam),
    );
    console.log("Success!", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考PutEvents中的。

以增量方式导入用户

以下代码示例展示了如何以增量方式将用户导入 Amazon Personalize Events 事件。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 [Amazon 代码示例存储库](https://github.com/awsdocs/aws-doc-sdk-examples)中查找完整示例，了解如何进行设置和运行。

```
// Get service clients module and commands using ES6 syntax.
import { PutUsersCommand } from "@aws-sdk/client-personalize-events";
import { personalizeEventsClient } from "./libs/personalizeClients.js";
// Or, create the client here.
// const personalizeEventsClient = new PersonalizeEventsClient({ region: "REGION"});

// Set the put users parameters. For string properties and values, use the \
 character to escape quotes.
var putUsersParam = {
  datasetArn: "DATASET_ARN",
  users: [
    {
      userId: "USER_ID",
      properties: '{"PROPERTY1_NAME": "PROPERTY1_VALUE"}',
    },
  ],
};
export const run = async () => {
  try {
    const response = await personalizeEventsClient.send(
      new PutUsersCommand(putUsersParam),
    );
    console.log("Success!", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考PutUsers中的。

Amazon 使用适用于 JavaScript (v3) 的软件开发工具包对运行时进行个性化示例

以下代码示例向您展示了如何使用带有 Amazon Personalize Runtime 的 Amazon SDK for JavaScript (v3) 来执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景 是展示如何通过在同一服务中调用多个函数来完成特定任务的代码示例。

每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关如何在上下文中设置和运行代码的说明。

主题

- 操作

操作

获取推荐（自定义数据集组）

以下代码示例展示了如何获取 Amazon Personalize Runtime 排名推荐。

适用于 JavaScript (v3) 的软件开发工具包

Note

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
// Get service clients module and commands using ES6 syntax.
import { GetPersonalizedRankingCommand } from
  "@aws-sdk/client-personalize-runtime";
import { personalizeRuntimeClient } from "./libs/personalizeClients.js";
// Or, create the client here.
```

```
// const personalizeRuntimeClient = new PersonalizeRuntimeClient({ region:
 "REGION"});

// Set the ranking request parameters.
export const getPersonalizedRankingParam = {
  campaignArn: "CAMPAIGN_ARN", /* required */
  userId: 'USER_ID',      /* required */
  inputList: ["ITEM_ID_1", "ITEM_ID_2", "ITEM_ID_3", "ITEM_ID_4"]
}

export const run = async () => {
  try {
    const response = await personalizeRuntimeClient.send(new
 GetPersonalizedRankingCommand(getPersonalizedRankingParam));
    console.log("Success!", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考GetPersonalizedRanking中的。

从推荐系统（域数据集组）处获取推荐

以下代码示例展示了如何获取 Amazon Personalize Runtime 推荐。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
// Get service clients module and commands using ES6 syntax.
import { GetRecommendationsCommand } from
```

```
  "@aws-sdk/client-personalize-runtime";

import { personalizeRuntimeClient } from "./libs/personalizeClients.js";
// Or, create the client here.
// const personalizeRuntimeClient = new PersonalizeRuntimeClient({ region:
 "REGION"});

// Set the recommendation request parameters.
export const getRecommendationsParam = {
  campaignArn: 'CAMPAIGN_ARN', /* required */
  userId: 'USER_ID',      /* required */
  numResults: 15    /* optional */
}

export const run = async () => {
  try {
    const response = await personalizeRuntimeClient.send(new
 GetRecommendationsCommand(getRecommendationsParam));
    console.log("Success!", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

使用筛选条件（自定义数据集组）获取推荐。

```
// Get service clients module and commands using ES6 syntax.
import { GetRecommendationsCommand } from
  "@aws-sdk/client-personalize-runtime";
import { personalizeRuntimeClient } from "./libs/personalizeClients.js";
// Or, create the client here.
// const personalizeRuntimeClient = new PersonalizeRuntimeClient({ region:
 "REGION"});

// Set the recommendation request parameters.
export const getRecommendationsParam = {
  recommenderArn: 'RECOMMENDER_ARN', /* required */
  userId: 'USER_ID',       /* required */
  numResults: 15    /* optional */
```

```
}

export const run = async () => {
  try {
    const response = await personalizeRuntimeClient.send(new
 GetRecommendationsCommand(getRecommendationsParam));
    console.log("Success!", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

从在域数据集组中创建的推荐系统处获取经过筛选的推荐。

```
// Get service clients module and commands using ES6 syntax.
import { GetRecommendationsCommand } from
  "@aws-sdk/client-personalize-runtime";
import { personalizeRuntimeClient } from "./libs/personalizeClients.js";
// Or, create the client here:
// const personalizeRuntimeClient = new PersonalizeRuntimeClient({ region:
 "REGION"});

// Set recommendation request parameters.
export const getRecommendationsParam = {
  campaignArn: 'CAMPAIGN_ARN', /* required */
  userId: 'USER_ID',      /* required */
  numResults: 15,    /* optional */
  filterArn: 'FILTER_ARN',   /* required to filter recommendations */
  filterValues: {
    "PROPERTY": "\"VALUE\""  /* Only required if your filter has a placeholder
 parameter */
  }
}

export const run = async () => {
  try {
    const response = await personalizeRuntimeClient.send(new
 GetRecommendationsCommand(getRecommendationsParam));
    console.log("Success!", response);
    return response; // For unit tests.
```

```
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考GetRecommendations中的。

使用适用于 JavaScript (v3) 的软件开发工具包的亚马逊 Pinpoint 示例

以下代码示例向您展示了如何使用带有 Amazon Pinpoint 的 Amazon SDK for JavaScript (v3) 来执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景 是展示如何通过在同一服务中调用多个函数来完成特定任务的代码示例。

每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关如何在上下文中设置和运行代码的说明。

主题

- 操作

操作

发送电子邮件和短信

以下代码示例展示了如何通过 Amazon Pinpoint 发送电子邮件和短信。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

在单独的模块中创建客户端并将其导出。

```
import { PinpointClient } from "@aws-sdk/client-pinpoint";
// Set the AWS Region.
const REGION = "us-east-1";
//Set the MediaConvert Service Object
const pinClient = new PinpointClient({ region: REGION });
export { pinClient };
```

发送电子邮件。

```
// Import required AWS SDK clients and commands for Node.js
import { SendMessagesCommand } from "@aws-sdk/client-pinpoint";
import { pinClient } from "./libs/pinClient.js";

// The FromAddress must be verified in SES.
const fromAddress = "FROM_ADDRESS";
const toAddress = "TO_ADDRESS";
const projectId = "PINPOINT_PROJECT_ID";

// The subject line of the email.
var subject = "Amazon Pinpoint Test (AWS SDK for JavaScript in Node.js)";

// The email body for recipients with non-HTML email clients.
var body_text = `Amazon Pinpoint Test (SDK for JavaScript in Node.js)
----------------------------------------------------
This email was sent with Amazon Pinpoint using the AWS SDK for JavaScript in
 Node.js.
For more information, see https://aws.amazon.com/sdk-for-node-js/`;

// The body of the email for recipients whose email clients support HTML content.
var body_html = `<html>
<head></head>
<body>
  <h1>Amazon Pinpoint Test (SDK for JavaScript in Node.js)</h1>
  <p>This email was sent with
    <a href='https://aws.amazon.com/pinpoint/'>the Amazon Pinpoint Email API</a>
 using the
    <a href='https://aws.amazon.com/sdk-for-node-js/'>
      AWS SDK for JavaScript in Node.js</a>.</p>
</body>
</html>`;
```

```
// The character encoding for the subject line and message body of the email.
var charset = "UTF-8";

const params = {
  ApplicationId: projectId,
  MessageRequest: {
    Addresses: {
      [toAddress]: {
        ChannelType: "EMAIL",
      },
    },
    MessageConfiguration: {
      EmailMessage: {
        FromAddress: fromAddress,
        SimpleEmail: {
          Subject: {
            Charset: charset,
            Data: subject,
          },
          HtmlPart: {
            Charset: charset,
            Data: body_html,
          },
          TextPart: {
            Charset: charset,
            Data: body_text,
          },
        },
      },
    },
  },
};

const run = async () => {
  try {
    const data = await pinClient.send(new SendMessagesCommand(params));

    const {
      MessageResponse: { Result },
    } = data;

    const recipientResult = Result[toAddress];
```

```
    if (recipientResult.StatusCode !== 200) {
      throw new Error(recipientResult.StatusMessage);
    } else {
      console.log(recipientResult.MessageId);
    }
  } catch (err) {
    console.log(err.message);
  }
};

run();
```

发送短信。

```
// Import required AWS SDK clients and commands for Node.js
import { SendMessagesCommand } from "@aws-sdk/client-pinpoint";
import { pinClient } from "./libs/pinClient.js";

("use strict");

/* The phone number or short code to send the message from. The phone number
 or short code that you specify has to be associated with your Amazon Pinpoint
account. For best results, specify long codes in E.164 format. */
const originationNumber = "SENDER_NUMBER"; //e.g., +1XXXXXXXXXX

// The recipient's phone number.  For best results, you should specify the phone
 number in E.164 format.
const destinationNumber = "RECEIVER_NUMBER"; //e.g., +1XXXXXXXXXX

// The content of the SMS message.
const message =
  "This message was sent through Amazon Pinpoint " +
  "using the AWS SDK for JavaScript in Node.js. Reply STOP to " +
  "opt out.";

/*The Amazon Pinpoint project/application ID to use when you send this message.
Make sure that the SMS channel is enabled for the project or application
that you choose.*/
const projectId = "PINPOINT_PROJECT_ID"; //e.g., XXXXXXXX66e4e9986478cXXXXXXXXX

/* The type of SMS message that you want to send. If you plan to send
```

```
time-sensitive content, specify TRANSACTIONAL. If you plan to send
marketing-related content, specify PROMOTIONAL.*/
var messageType = "TRANSACTIONAL";

// The registered keyword associated with the originating short code.
var registeredKeyword = "myKeyword";

/* The sender ID to use when sending the message. Support for sender ID
// varies by country or region. For more information, see
https://docs.aws.amazon.com/pinpoint/latest/userguide/channels-sms-countries.html.*/

var senderId = "MySenderID";

// Specify the parameters to pass to the API.
var params = {
  ApplicationId: projectId,
  MessageRequest: {
    Addresses: {
      [destinationNumber]: {
        ChannelType: "SMS",
      },
    },
    MessageConfiguration: {
      SMSMessage: {
        Body: message,
        Keyword: registeredKeyword,
        MessageType: messageType,
        OriginationNumber: originationNumber,
        SenderId: senderId,
      },
    },
  },
};

const run = async () => {
  try {
    const data = await pinClient.send(new SendMessagesCommand(params));
    return data; // For unit tests.
    console.log(
      "Message sent! " +
        data["MessageResponse"]["Result"][destinationNumber]["StatusMessage"]
    );
  } catch (err) {
    console.log(err);
```

```
  }
};
run();
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考SendMessages中的。

适用于 JavaScript (v2) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

发送电子邮件。

```
"use strict";

const AWS = require("aws-sdk");

// The AWS Region that you want to use to send the email. For a list of
// AWS Regions where the Amazon Pinpoint API is available, see
// https://docs.aws.amazon.com/pinpoint/latest/apireference/
const aws_region = "us-west-2";

// The "From" address. This address has to be verified in Amazon Pinpoint
// in the region that you use to send email.
const senderAddress = "sender@example.com";

// The address on the "To" line. If your Amazon Pinpoint account is in
// the sandbox, this address also has to be verified.
var toAddress = "recipient@example.com";

// The Amazon Pinpoint project/application ID to use when you send this message.
// Make sure that the SMS channel is enabled for the project or application
// that you choose.
const appId = "ce796be37f32f178af652b26eexample";

// The subject line of the email.
var subject = "Amazon Pinpoint (AWS SDK for JavaScript in Node.js)";
```

```
// The email body for recipients with non-HTML email clients.
var body_text = `Amazon Pinpoint Test (SDK for JavaScript in Node.js)
----------------------------------------------------
This email was sent with Amazon Pinpoint using the AWS SDK for JavaScript in
 Node.js.
For more information, see https:\/\/aws.amazon.com/sdk-for-node-js/`;

// The body of the email for recipients whose email clients support HTML content.
var body_html = `<html>
<head></head>
<body>
  <h1>Amazon Pinpoint Test (SDK for JavaScript in Node.js)</h1>
  <p>This email was sent with
    <a href='https://aws.amazon.com/pinpoint/'>the Amazon Pinpoint API</a> using the
    <a href='https://aws.amazon.com/sdk-for-node-js/'>
      AWS SDK for JavaScript in Node.js</a>.</p>
</body>
</html>`;

// The character encoding the you want to use for the subject line and
// message body of the email.
var charset = "UTF-8";

// Specify that you're using a shared credentials file.
var credentials = new AWS.SharedIniFileCredentials({ profile: "default" });
AWS.config.credentials = credentials;

// Specify the region.
AWS.config.update({ region: aws_region });

//Create a new Pinpoint object.
var pinpoint = new AWS.Pinpoint();

// Specify the parameters to pass to the API.
var params = {
  ApplicationId: appId,
  MessageRequest: {
    Addresses: {
      [toAddress]: {
        ChannelType: "EMAIL",
      },
    },
    MessageConfiguration: {
      EmailMessage: {
```

```
        FromAddress: senderAddress,
        SimpleEmail: {
          Subject: {
            Charset: charset,
            Data: subject,
          },
          HtmlPart: {
            Charset: charset,
            Data: body_html,
          },
          TextPart: {
            Charset: charset,
            Data: body_text,
          },
        },
      },
    },
  },
};

//Try to send the email.
pinpoint.sendMessages(params, function (err, data) {
  // If something goes wrong, print an error message.
  if (err) {
    console.log(err.message);
  } else {
    console.log(
      "Email sent! Message ID: ",
      data["MessageResponse"]["Result"][toAddress]["MessageId"]
    );
  }
});
```

发送短信。

```
"use strict";

var AWS = require("aws-sdk");

// The AWS Region that you want to use to send the message. For a list of
```

```
// AWS Regions where the Amazon Pinpoint API is available, see
// https://docs.aws.amazon.com/pinpoint/latest/apireference/.
var aws_region = "us-east-1";

// The phone number or short code to send the message from. The phone number
// or short code that you specify has to be associated with your Amazon Pinpoint
// account. For best results, specify long codes in E.164 format.
var originationNumber = "+12065550199";

// The recipient's phone number.  For best results, you should specify the
// phone number in E.164 format.
var destinationNumber = "+14255550142";

// The content of the SMS message.
var message =
  "This message was sent through Amazon Pinpoint " +
  "using the AWS SDK for JavaScript in Node.js. Reply STOP to " +
  "opt out.";

// The Amazon Pinpoint project/application ID to use when you send this message.
// Make sure that the SMS channel is enabled for the project or application
// that you choose.
var applicationId = "ce796be37f32f178af652b26eexample";

// The type of SMS message that you want to send. If you plan to send
// time-sensitive content, specify TRANSACTIONAL. If you plan to send
// marketing-related content, specify PROMOTIONAL.
var messageType = "TRANSACTIONAL";

// The registered keyword associated with the originating short code.
var registeredKeyword = "myKeyword";

// The sender ID to use when sending the message. Support for sender ID
// varies by country or region. For more information, see
// https://docs.aws.amazon.com/pinpoint/latest/userguide/channels-sms-countries.html
var senderId = "MySenderID";

// Specify that you're using a shared credentials file, and optionally specify
// the profile that you want to use.
var credentials = new AWS.SharedIniFileCredentials({ profile: "default" });
AWS.config.credentials = credentials;

// Specify the region.
AWS.config.update({ region: aws_region });
```

```
//Create a new Pinpoint object.
var pinpoint = new AWS.Pinpoint();

// Specify the parameters to pass to the API.
var params = {
  ApplicationId: applicationId,
  MessageRequest: {
    Addresses: {
      [destinationNumber]: {
        ChannelType: "SMS",
      },
    },
    MessageConfiguration: {
      SMSMessage: {
        Body: message,
        Keyword: registeredKeyword,
        MessageType: messageType,
        OriginationNumber: originationNumber,
        SenderId: senderId,
      },
    },
  },
};

//Try to send the message.
pinpoint.sendMessages(params, function (err, data) {
  // If something goes wrong, print an error message.
  if (err) {
    console.log(err.message);
    // Otherwise, show the unique ID for the message.
  } else {
    console.log(
      "Message sent! " +
        data["MessageResponse"]["Result"][destinationNumber]["StatusMessage"]
    );
  }
});
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考SendMessages中的。

使用适用于 JavaScript (v3) 的软件开发工具包的亚马逊 Redshift 示例

以下代码示例向您展示了如何使用带有 Amazon Redshift 的 Amazon SDK for JavaScript (v3) 来执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景 是展示如何通过在同一服务中调用多个函数来完成特定任务的代码示例。

每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关如何在上下文中设置和运行代码的说明。

主题

- 操作

操作

创建集群

以下代码示例展示了如何创建 Amazon Redshift 集群。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。查找完整示例，了解如何在 Amazon 代码示例存储库中进行设置和运行。

创建客户端。

```
const { RedshiftClient } = require("@aws-sdk/client-redshift");
// Set the AWS Region.
const REGION = "REGION";
//Set the Redshift Service Object
const redshiftClient = new RedshiftClient({ region: REGION });
export { redshiftClient };
```

创建集群。

```
// Import required AWS SDK clients and commands for Node.js
import { CreateClusterCommand } from "@aws-sdk/client-redshift";
import { redshiftClient } from "./libs/redshiftClient.js";

const params = {
  ClusterIdentifier: "CLUSTER_NAME", // Required
  NodeType: "NODE_TYPE", //Required
  MasterUsername: "MASTER_USER_NAME", // Required - must be lowercase
  MasterUserPassword: "MASTER_USER_PASSWORD", // Required - must contain at least
 one uppercase letter, and one number
  ClusterType: "CLUSTER_TYPE", // Required
  IAMRoleARN: "IAM_ROLE_ARN", // Optional - the ARN of an IAM role with permissions
 your cluster needs to access other AWS services on your behalf, such as Amazon S3.
  ClusterSubnetGroupName: "CLUSTER_SUBNET_GROUPNAME", //Optional - the name of a
 cluster subnet group to be associated with this cluster. Defaults to 'default' if
 not specified.
  DBName: "DATABASE_NAME", // Optional - defaults to 'dev' if not specified
  Port: "PORT_NUMBER", // Optional - defaults to '5439' if not specified
};

const run = async () => {
  try {
    const data = await redshiftClient.send(new CreateClusterCommand(params));
    console.log(
      "Cluster " + data.Cluster.ClusterIdentifier + " successfully created",
    );
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考CreateCluster中的。

删除集群

以下代码示例展示了如何删除 Amazon Redshift 集群。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。查找完整示例，了解如何在 Amazon 代码示例存储库中进行设置和运行。

创建客户端。

```
const { RedshiftClient } = require("@aws-sdk/client-redshift");
// Set the AWS Region.
const REGION = "REGION";
//Set the Redshift Service Object
const redshiftClient = new RedshiftClient({ region: REGION });
export { redshiftClient };
```

创建集群。

```
// Import required AWS SDK clients and commands for Node.js
import { DeleteClusterCommand } from "@aws-sdk/client-redshift";
import { redshiftClient } from "./libs/redshiftClient.js";

const params = {
  ClusterIdentifier: "CLUSTER_NAME",
  SkipFinalClusterSnapshot: false,
  FinalClusterSnapshotIdentifier: "CLUSTER_SNAPSHOT_ID",
};

const run = async () => {
  try {
    const data = await redshiftClient.send(new DeleteClusterCommand(params));
    console.log("Success, cluster deleted. ", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DeleteCluster中的。

描述集群

以下代码示例展示了如何描述 Amazon Redshift 集群。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。查找完整示例，了解如何在 Amazon 代码示例存储库中进行设置和运行。

创建客户端。

```
const { RedshiftClient } = require("@aws-sdk/client-redshift");
// Set the AWS Region.
const REGION = "REGION";
//Set the Redshift Service Object
const redshiftClient = new RedshiftClient({ region: REGION });
export { redshiftClient };
```

描述集群。

```
// Import required AWS SDK clients and commands for Node.js
import { DescribeClustersCommand } from "@aws-sdk/client-redshift";
import { redshiftClient } from "./libs/redshiftClient.js";

const params = {
  ClusterIdentifier: "CLUSTER_NAME",
};

const run = async () => {
  try {
    const data = await redshiftClient.send(new DescribeClustersCommand(params));
    console.log("Success", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
```

```
  }
};
run();
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DescribeClusters中的。

修改集群

以下代码示例展示了如何修改 Amazon Redshift 集群。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。查找完整示例，了解如何在 Amazon 代码示例存储库中进行设置和运行。

创建客户端。

```
const { RedshiftClient } = require("@aws-sdk/client-redshift");
// Set the AWS Region.
const REGION = "REGION";
//Set the Redshift Service Object
const redshiftClient = new RedshiftClient({ region: REGION });
export { redshiftClient };
```

修改集群。

```
// Import required AWS SDK clients and commands for Node.js
import { ModifyClusterCommand } from "@aws-sdk/client-redshift";
import { redshiftClient } from "./libs/redshiftClient.js";

// Set the parameters
const params = {
  ClusterIdentifier: "CLUSTER_NAME",
  MasterUserPassword: "NEW_MASTER_USER_PASSWORD",
};
```

```
const run = async () => {
  try {
    const data = await redshiftClient.send(new ModifyClusterCommand(params));
    console.log("Success was modified.", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考ModifyCluster中的。

使用适用于 JavaScript (v3) 的软件开发工具包的 Amazon S3 示例

以下代码示例向您展示了如何在 Amazon S3 中使用 Amazon SDK for JavaScript (v3) 来执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景 是展示如何通过在同一服务中调用多个函数来完成特定任务的代码示例。

每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关如何在上下文中设置和运行代码的说明。

开始使用

开始使用 Amazon S3

以下代码示例展示了如何开始使用 Amazon S3。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { ListBucketsCommand, S3Client } from "@aws-sdk/client-s3";

// When no region or credentials are provided, the SDK will use the
// region and credentials from the local AWS config.
const client = new S3Client({});

export const helloS3 = async () => {
  const command = new ListBucketsCommand({});

  const { Buckets } = await client.send(command);
  console.log("Buckets: ");
  console.log(Buckets.map((bucket) => bucket.Name).join("\n"));
  return Buckets;
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考ListBuckets中的。

主题

- 操作
- 场景

操作

将 CORS 规则添加到桶

以下代码示例展示了如何向 S3 存储桶添加跨源资源共享 (CORS) 规则。

适用于 JavaScript (v3) 的软件开发工具包

Note

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

添加 CORS 规则。

```
import { PutBucketCorsCommand, S3Client } from "@aws-sdk/client-s3";
```

```
const client = new S3Client({});

// By default, Amazon S3 doesn't allow cross-origin requests. Use this command
// to explicitly allow cross-origin requests.
export const main = async () => {
  const command = new PutBucketCorsCommand({
    Bucket: "test-bucket",
    CORSConfiguration: {
      CORSRules: [
        {
          // Allow all headers to be sent to this bucket.
          AllowedHeaders: ["*"],
          // Allow only GET and PUT methods to be sent to this bucket.
          AllowedMethods: ["GET", "PUT"],
          // Allow only requests from the specified origin.
          AllowedOrigins: ["https://www.example.com"],
          // Allow the entity tag (ETag) header to be returned in the response. The
 ETag header
          // The entity tag represents a specific version of the object. The ETag
 reflects
          // changes only to the contents of an object, not its metadata.
          ExposeHeaders: ["ETag"],
          // How long the requesting browser should cache the preflight response.
 After
          // this time, the preflight request will have to be made again.
          MaxAgeSeconds: 3600,
        },
      ],
    },
  });

  try {
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考PutBucketCors中的。

将策略添加到桶

以下代码示例展示了如何将策略添加到 S3 存储桶。

适用于 JavaScript (v3) 的软件开发工具包

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

添加策略。

```
import { PutBucketPolicyCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

export const main = async () => {
  const command = new PutBucketPolicyCommand({
    Policy: JSON.stringify({
      Version: "2012-10-17",
      Statement: [
        {
          Sid: "AllowGetObject",
          // Allow this particular user to call GetObject on any object in this
 bucket.
          Effect: "Allow",
          Principal: {
            AWS: "arn:aws:iam::ACCOUNT-ID:user/USERNAME",
          },
          Action: "s3:GetObject",
          Resource: "arn:aws:s3:::BUCKET-NAME/*",
        },
      ],
    }),
    // Apply the preceding policy to this bucket.
    Bucket: "BUCKET-NAME",
  });

  try {
    const response = await client.send(command);
    console.log(response);
```

```
  } catch (err) {
    console.error(err);
  }
};
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考PutBucketPolicy中的。

将对象从一个桶复制到另一个桶

以下代码示例展示了如何将 S3 对象从一个桶复制到另一个桶。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

复制对象。

```
import { S3Client, CopyObjectCommand } from "@aws-sdk/client-s3";

const client = new S3Client({});

export const main = async () => {
  const command = new CopyObjectCommand({
    CopySource: "SOURCE_BUCKET/SOURCE_OBJECT_KEY",
    Bucket: "DESTINATION_BUCKET",
    Key: "NEW_OBJECT_KEY",
  });

  try {
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考CopyObject中的。

创建桶

以下代码示例展示了如何创建 S3 桶。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

创建存储桶。

```
import { CreateBucketCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

export const main = async () => {
  const command = new CreateBucketCommand({
    // The name of the bucket. Bucket names are unique and have several other
 constraints.
    // See https://docs.aws.amazon.com/AmazonS3/latest/userguide/
bucketnamingrules.html
    Bucket: "bucket-name",
  });

  try {
    const { Location } = await client.send(command);
    console.log(`Bucket created with location ${Location}`);
  } catch (err) {
    console.error(err);
  }
};
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考CreateBucket中的。

从存储桶中删除策略

以下代码示例展示了如何从 S3 存储桶中删除策略。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

删除存储桶策略。

```
import { DeleteBucketPolicyCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

// This will remove the policy from the bucket.
export const main = async () => {
  const command = new DeleteBucketPolicyCommand({
    Bucket: "test-bucket",
  });

  try {
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DeleteBucketPolicy中的。

删除空存储桶

以下代码示例展示了如何删除空的 S3 桶。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

删除存储桶。

```
import { DeleteBucketCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

// Delete a bucket.
export const main = async () => {
  const command = new DeleteBucketCommand({
    Bucket: "test-bucket",
  });

  try {
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DeleteBucket中的。

删除对象

以下代码示例展示了如何删除 S3 对象。

适用于 JavaScript (v3) 的软件开发工具包

Note

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

删除对象。

```
import { DeleteObjectCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

export const main = async () => {
  const command = new DeleteObjectCommand({
    Bucket: "test-bucket",
    Key: "test-key.txt",
  });

  try {
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DeleteObject中的。

删除多个对象

以下代码示例展示了如何从 S3 桶中删除多个对象。

适用于 JavaScript (v3) 的软件开发工具包

Note

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

删除多个对象。

```
import { DeleteObjectsCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

export const main = async () => {
  const command = new DeleteObjectsCommand({
    Bucket: "test-bucket",
    Delete: {
      Objects: [{ Key: "object1.txt" }, { Key: "object2.txt" }],
    },
  });

  try {
    const { Deleted } = await client.send(command);
    console.log(
      `Successfully deleted ${Deleted.length} objects from S3 bucket. Deleted
 objects:`,
    );
    console.log(Deleted.map((d) => ` • ${d.Key}`).join("\n"));
  } catch (err) {
    console.error(err);
  }
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DeleteObjects中的。

删除存储桶的网站配置

以下代码示例展示了如何从 S3 存储桶中删除网站配置。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

从存储桶中删除网站配置。

```
import { DeleteBucketWebsiteCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

// Disable static website hosting on the bucket.
export const main = async () => {
  const command = new DeleteBucketWebsiteCommand({
    Bucket: "test-bucket",
  });

  try {
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DeleteBucketWebsite中的。

获取存储桶的 CORS 规则

以下代码示例展示了如何获取 S3 存储桶的跨源资源共享 (CORS) 规则。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

获取存储桶的 CORS 策略。

```
import { GetBucketCorsCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});
```

```
export const main = async () => {
  const command = new GetBucketCorsCommand({
    Bucket: "test-bucket",
  });

  try {
    const { CORSRules } = await client.send(command);
    CORSRules.forEach((cr, i) => {
      console.log(
        `\nCORSRule ${i + 1}`,
        `\n${"-".repeat(10)}`,
        `\nAllowedHeaders: ${cr.AllowedHeaders.join(" ")}`,
        `\nAllowedMethods: ${cr.AllowedMethods.join(" ")}`,
        `\nAllowedOrigins: ${cr.AllowedOrigins.join(" ")}`,
        `\nExposeHeaders: ${cr.ExposeHeaders.join(" ")}`,
        `\nMaxAgeSeconds: ${cr.MaxAgeSeconds}`,
      );
    });
  } catch (err) {
    console.error(err);
  }
};
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考GetBucketCors中的。

从存储桶中获取对象

以下代码示例展示了如何从 S3 桶中的对象读取数据。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

下载对象。

```
import { GetObjectCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

export const main = async () => {
  const command = new GetObjectCommand({
    Bucket: "test-bucket",
    Key: "hello-s3.txt",
  });

  try {
    const response = await client.send(command);
    // The Body object also has 'transformToByteArray' and 'transformToWebStream'
 methods.
    const str = await response.Body.transformToString();
    console.log(str);
  } catch (err) {
    console.error(err);
  }
};
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考GetObject中的。

获取存储桶的 ACL

以下代码示例展示了如何获取 S3 桶的访问控制列表（ACL）。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

获取 ACL 权限。

```
import { GetBucketAclCommand, S3Client } from "@aws-sdk/client-s3";
```

```
const client = new S3Client({});

export const main = async () => {
  const command = new GetBucketAclCommand({
    Bucket: "test-bucket",
  });

  try {
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考GetBucketAcl中的。

获取存储桶的策略

以下代码示例展示了如何获取 S3 存储桶的策略。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

获取存储桶策略。

```
import { GetBucketPolicyCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

export const main = async () => {
  const command = new GetBucketPolicyCommand({
    Bucket: "test-bucket",
```

```
  });

  try {
    const { Policy } = await client.send(command);
    console.log(JSON.parse(Policy));
  } catch (err) {
    console.error(err);
  }
};
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考GetBucketPolicy中的。

获取存储桶网站的配置

以下代码示例展示了如何获取 S3 桶的网站配置。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

获取网站配置。

```
import { GetBucketWebsiteCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

export const main = async () => {
  const command = new GetBucketWebsiteCommand({
    Bucket: "test-bucket",
  });

  try {
    const { ErrorDocument, IndexDocument } = await client.send(command);
    console.log(
      `Your bucket is set up to host a website. It has an error document:`,
```

```
      `${ErrorDocument.Key}, and an index document: ${IndexDocument.Suffix}.`,
    );
  } catch (err) {
    console.error(err);
  }
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考GetBucketWebsite中的。

列出存储桶

以下代码示例展示了如何列出 S3 存储桶。

适用于 JavaScript (v3) 的软件开发工具包

> Note
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

列出存储桶。

```
import { ListBucketsCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

export const main = async () => {
  const command = new ListBucketsCommand({});

  try {
    const { Owner, Buckets } = await client.send(command);
    console.log(
      `${Owner.DisplayName} owns ${Buckets.length} bucket${
        Buckets.length === 1 ? "" : "s"
      }:`,
    );
    console.log(`${Buckets.map((b) => ` • ${b.Name}`).join("\n")}`);
  } catch (err) {
    console.error(err);
  }
```

```
};
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考ListBuckets中的。

列出存储桶中的对象

以下代码示例展示了如何列出 S3 桶中的对象。

适用于 JavaScript (v3) 的软件开发工具包

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

列出存储桶中的所有对象。如果有多个对象，则 NextContinuationToken 将使用 IsTruncated 并遍历完整列表。

```
import {
  S3Client,
  // This command supersedes the ListObjectsCommand and is the recommended way to
 list objects.
  ListObjectsV2Command,
} from "@aws-sdk/client-s3";

const client = new S3Client({});

export const main = async () => {
  const command = new ListObjectsV2Command({
    Bucket: "my-bucket",
    // The default and maximum number of keys returned is 1000. This limits it to
    // one for demonstration purposes.
    MaxKeys: 1,
  });

  try {
    let isTruncated = true;
```

```
    console.log("Your bucket contains the following objects:\n");
    let contents = "";

    while (isTruncated) {
      const { Contents, IsTruncated, NextContinuationToken } =
        await client.send(command);
      const contentsList = Contents.map((c) => ` • ${c.Key}`).join("\n");
      contents += contentsList + "\n";
      isTruncated = IsTruncated;
      command.input.ContinuationToken = NextContinuationToken;
    }
    console.log(contents);
  } catch (err) {
    console.error(err);
  }
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考中的 ListObjectsV2。

为存储桶设置新 ACL

以下代码示例展示了如何为 S3 桶设置新的访问控制列表（ACL）。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

放置存储桶 ACL。

```
import {
  PutBucketAclCommand,
  GetBucketAclCommand,
  S3Client,
} from "@aws-sdk/client-s3";

const client = new S3Client({});
```

```
// Most Amazon S3 use cases don't require the use of access control lists (ACLs).
// We recommend that you disable ACLs, except in unusual circumstances where
// you need to control access for each object individually.
// Consider a policy instead. For more information see https://docs.aws.amazon.com/
AmazonS3/latest/userguide/bucket-policies.html.
export const main = async () => {
  // Grant a user READ access to a bucket.
  const command = new PutBucketAclCommand({
    Bucket: "test-bucket",
    AccessControlPolicy: {
      Grants: [
        {
          Grantee: {
            // The canonical ID of the user. This ID is an obfuscated form of your
 AWS account number.
            // It's unique to Amazon S3 and can't be found elsewhere.
            // For more information, see https://docs.aws.amazon.com/AmazonS3/
latest/userguide/finding-canonical-user-id.html.
            ID: "canonical-id-1",
            Type: "CanonicalUser",
          },
          // One of FULL_CONTROL | READ | WRITE | READ_ACP | WRITE_ACP
          // https://docs.aws.amazon.com/AmazonS3/latest/API/
API_Grant.html#AmazonS3-Type-Grant-Permission
          Permission: "FULL_CONTROL",
        },
      ],
      Owner: {
        ID: "canonical-id-2",
      },
    },
  });

  try {
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考PutBucketAcl中的。

设置存储桶的网站配置

以下示例展示了如何设置 S3 存储桶的网站配置。

适用于 JavaScript (v3) 的软件开发工具包

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

设置网站配置。

```
import { PutBucketWebsiteCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

// Set up a bucket as a static website.
// The bucket needs to be publicly accessible.
export const main = async () => {
  const command = new PutBucketWebsiteCommand({
    Bucket: "test-bucket",
    WebsiteConfiguration: {
      ErrorDocument: {
        // The object key name to use when a 4XX class error occurs.
        Key: "error.html",
      },
      IndexDocument: {
        // A suffix that is appended to a request that is for a directory.
        Suffix: "index.html",
      },
    },
  });

  try {
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
```

```
  }
};
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考PutBucketWebsite中的。

上传对象到存储桶

以下代码示例展示了如何将对象上传到 S3 桶。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

上传对象。

```
import { PutObjectCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

export const main = async () => {
  const command = new PutObjectCommand({
    Bucket: "test-bucket",
    Key: "hello-s3.txt",
    Body: "Hello S3!",
  });

  try {
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考PutObject中的。

场景

创建预签名 URL

以下代码示例展示了如何为 Amazon S3 创建预签名 URL 以及如何上传对象。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

创建预签名 URL 以将对象上传到存储桶。

```
import https from "https";
import { PutObjectCommand, S3Client } from "@aws-sdk/client-s3";
import { fromIni } from "@aws-sdk/credential-providers";
import { HttpRequest } from "@smithy/protocol-http";
import {
  getSignedUrl,
  S3RequestPresigner,
} from "@aws-sdk/s3-request-presigner";
import { parseUrl } from "@smithy/url-parser";
import { formatUrl } from "@aws-sdk/util-format-url";
import { Hash } from "@smithy/hash-node";

const createPresignedUrlWithoutClient = async ({ region, bucket, key }) => {
  const url = parseUrl(`https://${bucket}.s3.${region}.amazonaws.com/${key}`);
  const presigner = new S3RequestPresigner({
    credentials: fromIni(),
    region,
    sha256: Hash.bind(null, "sha256"),
  });

  const signedUrlObject = await presigner.presign(
    new HttpRequest({ ...url, method: "PUT" }),
  );
```

```
  return formatUrl(signedUrlObject);
};

const createPresignedUrlWithClient = ({ region, bucket, key }) => {
  const client = new S3Client({ region });
  const command = new PutObjectCommand({ Bucket: bucket, Key: key });
  return getSignedUrl(client, command, { expiresIn: 3600 });
};

function put(url, data) {
  return new Promise((resolve, reject) => {
    const req = https.request(
      url,
      { method: "PUT", headers: { "Content-Length": new Blob([data]).size } },
      (res) => {
        let responseBody = "";
        res.on("data", (chunk) => {
          responseBody += chunk;
        });
        res.on("end", () => {
          resolve(responseBody);
        });
      },
    );
    req.on("error", (err) => {
      reject(err);
    });
    req.write(data);
    req.end();
  });
}

export const main = async () => {
  const REGION = "us-east-1";
  const BUCKET = "example_bucket";
  const KEY = "example_file.txt";

  // There are two ways to generate a presigned URL.
  // 1. Use createPresignedUrl without the S3 client.
  // 2. Use getSignedUrl in conjunction with the S3 client and GetObjectCommand.
  try {
    const noClientUrl = await createPresignedUrlWithoutClient({
      region: REGION,
      bucket: BUCKET,
```

```
      key: KEY,
    });

    const clientUrl = await createPresignedUrlWithClient({
      region: REGION,
      bucket: BUCKET,
      key: KEY,
    });

    // After you get the presigned URL, you can provide your own file
    // data. Refer to put() above.
    console.log("Calling PUT using presigned URL without client");
    await put(noClientUrl, "Hello World");

    console.log("Calling PUT using presigned URL with client");
    await put(clientUrl, "Hello World");

    console.log("\nDone. Check your S3 console.");
  } catch (err) {
    console.error(err);
  }
};
```

创建预签名 URL 以从存储桶下载对象。

```
import { GetObjectCommand, S3Client } from "@aws-sdk/client-s3";
import { fromIni } from "@aws-sdk/credential-providers";
import { HttpRequest } from "@smithy/protocol-http";
import {
  getSignedUrl,
  S3RequestPresigner,
} from "@aws-sdk/s3-request-presigner";
import { parseUrl } from "@smithy/url-parser";
import { formatUrl } from "@aws-sdk/util-format-url";
import { Hash } from "@smithy/hash-node";

const createPresignedUrlWithoutClient = async ({ region, bucket, key }) => {
  const url = parseUrl(`https://${bucket}.s3.${region}.amazonaws.com/${key}`);
  const presigner = new S3RequestPresigner({
    credentials: fromIni(),
    region,
    sha256: Hash.bind(null, "sha256"),
```

```
  });

  const signedUrlObject = await presigner.presign(new HttpRequest(url));
  return formatUrl(signedUrlObject);
};

const createPresignedUrlWithClient = ({ region, bucket, key }) => {
  const client = new S3Client({ region });
  const command = new GetObjectCommand({ Bucket: bucket, Key: key });
  return getSignedUrl(client, command, { expiresIn: 3600 });
};

export const main = async () => {
  const REGION = "us-east-1";
  const BUCKET = "example_bucket";
  const KEY = "example_file.jpg";

  try {
    const noClientUrl = await createPresignedUrlWithoutClient({
      region: REGION,
      bucket: BUCKET,
      key: KEY,
    });

    const clientUrl = await createPresignedUrlWithClient({
      region: REGION,
      bucket: BUCKET,
      key: KEY,
    });

    console.log("Presigned URL without client");
    console.log(noClientUrl);
    console.log("\n");

    console.log("Presigned URL with client");
    console.log(clientUrl);
  } catch (err) {
    console.error(err);
  }
};
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。

创建列出 Amazon S3 对象的网页

以下代码示例展示了如何在网页中列出 Amazon S3 对象。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

以下代码是调用 Amazon SDK 的相关的 React 组件。可以在前面的 GitHub 链接中找到包含此组件的应用程序的可运行版本。

```
import { useEffect, useState } from "react";
import {
  ListObjectsCommand,
  ListObjectsCommandOutput,
  S3Client,
} from "@aws-sdk/client-s3";
import { fromCognitoIdentityPool } from "@aws-sdk/credential-providers";
import "./App.css";

function App() {
  const [objects, setObjects] = useState<
    Required<ListObjectsCommandOutput>["Contents"]
  >([]);

  useEffect(() => {
    const client = new S3Client({
      region: "us-east-1",
      // Unless you have a public bucket, you'll need access to a private bucket.
      // One way to do this is to create an Amazon Cognito identity pool, attach a
 role to the pool,
      // and grant the role access to the 's3:GetObject' action.
      //
      // You'll also need to configure the CORS settings on the bucket to allow
 traffic from
      // this example site. Here's an example configuration that allows all origins.
 Don't
      // do this in production.
```

```
      //[
      //  {
      //    "AllowedHeaders": ["*"],
      //    "AllowedMethods": ["GET"],
      //    "AllowedOrigins": ["*"],
      //    "ExposeHeaders": [],
      //  },
      //]
      //
      credentials: fromCognitoIdentityPool({
        clientConfig: { region: "us-east-1" },
        identityPoolId: "<YOUR_IDENTITY_POOL_ID>",
      }),
    });
    const command = new ListObjectsCommand({ Bucket: "bucket-name" });
    client.send(command).then(({ Contents }) => setObjects(Contents || []));
  }, []);

  return (
    <div className="App">
      {objects.map((o) => (
        <div key={o.ETag}>{o.Key}</div>
      ))}
    </div>
  );
}

export default App;
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考ListObjects中的。

桶和对象入门

以下代码示例展示了如何：

- 创建桶并将文件上载到其中。
- 从桶中下载对象。
- 将对象复制到存储桶中的子文件夹。
- 列出存储桶中的对象。
- 删除存储桶及其对象。

适用于 JavaScript (v3) 的软件开发工具包

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

首先，导入所有必需的模块。

```
// Used to check if currently running file is this file.
import { fileURLToPath } from "url";
import { readdirSync, readFileSync, writeFileSync } from "fs";

// Local helper utils.
import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";
import { Prompter } from "@aws-doc-sdk-examples/lib/prompter.js";
import { wrapText } from "@aws-doc-sdk-examples/lib/utils/util-string.js";

import {
  S3Client,
  CreateBucketCommand,
  PutObjectCommand,
  ListObjectsCommand,
  CopyObjectCommand,
  GetObjectCommand,
  DeleteObjectsCommand,
  DeleteBucketCommand,
} from "@aws-sdk/client-s3";
```

前面的导入引用了一些帮助程序实用程序。这些实用程序是本节开头链接的 GitHub 存储库的本地工具。为了便于参考，请参阅这些实用程序的以下实现。

```
export const dirnameFromMetaUrl = (metaUrl) =>
  fileURLToPath(new URL(".", metaUrl));

import { select, input, confirm, checkbox } from "@inquirer/prompts";

export class Prompter {
  /**
   * @param {{ message: string, choices: { name: string, value: string }[]}} options
```

```
   */
  select(options) {
    return select(options);
  }

  /**
   * @param {{ message: string }} options
   */
  input(options) {
    return input(options);
  }

  /**
   * @param {string} prompt
   */
  checkContinue = async (prompt = "") => {
    const prefix = prompt && prompt + " ";
    let ok = await this.confirm({
      message: `${prefix}Continue?`,
    });
    if (!ok) throw new Error("Exiting...");
  };

  /**
   * @param {{ message: string }} options
   */
  confirm(options) {
    return confirm(options);
  }

  /**
   * @param {{ message: string, choices: { name: string, value: string }[]}} options
   */
  checkbox(options) {
    return checkbox(options);
  }
}

export const wrapText = (text, char = "=") => {
  const rule = char.repeat(80);
  return `${rule}\n    ${text}\n${rule}\n`;
};
```

S3 中的对象存储在“存储桶”中。让我们定义一个用于创建新存储桶的函数。

```
export const createBucket = async () => {
  const bucketName = await prompter.input({
    message: "Enter a bucket name. Bucket names must be globally unique:",
  });
  const command = new CreateBucketCommand({ Bucket: bucketName });
  await s3Client.send(command);
  console.log("Bucket created successfully.\n");
  return bucketName;
};
```

存储桶包含“对象”。此函数将目录的内容作为对象上传到您的存储桶。

```
export const uploadFilesToBucket = async ({ bucketName, folderPath }) => {
  console.log(`Uploading files from ${folderPath}\n`);
  const keys = readdirSync(folderPath);
  const files = keys.map((key) => {
    const filePath = `${folderPath}/${key}`;
    const fileContent = readFileSync(filePath);
    return {
      Key: key,
      Body: fileContent,
    };
  });

  for (let file of files) {
    await s3Client.send(
      new PutObjectCommand({
        Bucket: bucketName,
        Body: file.Body,
        Key: file.Key,
      }),
    );
    console.log(`${file.Key} uploaded successfully.`);
  }
};
```

上传对象后，检查以确认它们已正确上传。你可以用来 ListObjects 做这个。您将使用“Key”属性，但响应中还有其他有用的属性。

```
export const listFilesInBucket = async ({ bucketName }) => {
  const command = new ListObjectsCommand({ Bucket: bucketName });
  const { Contents } = await s3Client.send(command);
  const contentsList = Contents.map((c) => ` • ${c.Key}`).join("\n");
  console.log("\nHere's a list of files in the bucket:");
  console.log(contentsList + "\n");
};
```

有时，您可能想要将对象从一个桶复制到另一个桶。使用 CopyObject 命令来做到这一点。

```
export const copyFileFromBucket = async ({ destinationBucket }) => {
  const proceed = await prompter.confirm({
    message: "Would you like to copy an object from another bucket?",
  });

  if (!proceed) {
    return;
  } else {
    const copy = async () => {
      try {
        const sourceBucket = await prompter.input({
          message: "Enter source bucket name:",
        });
        const sourceKey = await prompter.input({
          message: "Enter source key:",
        });
        const destinationKey = await prompter.input({
          message: "Enter destination key:",
        });

        const command = new CopyObjectCommand({
          Bucket: destinationBucket,
          CopySource: `${sourceBucket}/${sourceKey}`,
          Key: destinationKey,
        });
        await s3Client.send(command);
        await copyFileFromBucket({ destinationBucket });
      } catch (err) {
        console.error(`Copy error.`);
        console.error(err);
        const retryAnswer = await prompter.confirm({ message: "Try again?" });
        if (retryAnswer) {
```

```
          await copy();
        }
      }
    };
    await copy();
  }
};
```

没有用于从桶中获取多个对象的 SDK 方法。相反，您将创建一个要下载的对象列表并对其进行迭代。

```
export const downloadFilesFromBucket = async ({ bucketName }) => {
  const { Contents } = await s3Client.send(
    new ListObjectsCommand({ Bucket: bucketName }),
  );
  const path = await prompter.input({
    message: "Enter destination path for files:",
  });

  for (let content of Contents) {
    const obj = await s3Client.send(
      new GetObjectCommand({ Bucket: bucketName, Key: content.Key }),
    );
    writeFileSync(
      `${path}/${content.Key}`,
      await obj.Body.transformToByteArray(),
    );
  }
  console.log("Files downloaded successfully.\n");
};
```

是时候清除资源了。桶必须为空才能删除它。这两个函数清空并删除桶。

```
export const emptyBucket = async ({ bucketName }) => {
  const listObjectsCommand = new ListObjectsCommand({ Bucket: bucketName });
  const { Contents } = await s3Client.send(listObjectsCommand);
  const keys = Contents.map((c) => c.Key);

  const deleteObjectsCommand = new DeleteObjectsCommand({
    Bucket: bucketName,
    Delete: { Objects: keys.map((key) => ({ Key: key })) },
```

```
  });
  await s3Client.send(deleteObjectsCommand);
  console.log(`${bucketName} emptied successfully.\n`);
};

export const deleteBucket = async ({ bucketName }) => {
  const command = new DeleteBucketCommand({ Bucket: bucketName });
  await s3Client.send(command);
  console.log(`${bucketName} deleted successfully.\n`);
};
```

“main”函数将所有内容整合在一起。如果您直接运行此文件，将调用 main 函数。

```
const main = async () => {
  const OBJECT_DIRECTORY = `${dirnameFromMetaUrl(
    import.meta.url,
  )}../../../../resources/sample_files/.sample_media`;

  try {
    console.log(wrapText("Welcome to the Amazon S3 getting started example."));
    console.log("Let's create a bucket.");
    const bucketName = await createBucket();
    await prompter.confirm({ message: continueMessage });

    console.log(wrapText("File upload."));
    console.log(
      "I have some default files ready to go. You can edit the source code to
 provide your own.",
    );
    await uploadFilesToBucket({
      bucketName,
      folderPath: OBJECT_DIRECTORY,
    });

    await listFilesInBucket({ bucketName });
    await prompter.confirm({ message: continueMessage });

    console.log(wrapText("Copy files."));
    await copyFileFromBucket({ destinationBucket: bucketName });
    await listFilesInBucket({ bucketName });
    await prompter.confirm({ message: continueMessage });
```

```
    console.log(wrapText("Download files."));
    await downloadFilesFromBucket({ bucketName });

    console.log(wrapText("Clean up."));
    await emptyBucket({ bucketName });
    await deleteBucket({ bucketName });
  } catch (err) {
    console.error(err);
  }
};
```

- 有关 API 详细信息，请参阅《Amazon SDK for JavaScript API 参考》中的以下主题。
  - CopyObject
  - CreateBucket
  - DeleteBucket
  - DeleteObjects
  - GetObject
  - ListObjectsV2
  - PutObject

上传或下载大文件

下面的代码示例展示了如何向 Amazon S3 上传大文件或从 Amazon S3 下载大文件。

有关更多信息，请参阅使用分段上传操作上传对象。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

上传大文件。

```
import {
  CreateMultipartUploadCommand,
```

```
  UploadPartCommand,
  CompleteMultipartUploadCommand,
  AbortMultipartUploadCommand,
  S3Client,
} from "@aws-sdk/client-s3";

const twentyFiveMB = 25 * 1024 * 1024;

export const createString = (size = twentyFiveMB) => {
  return "x".repeat(size);
};

export const main = async () => {
  const s3Client = new S3Client({});
  const bucketName = "test-bucket";
  const key = "multipart.txt";
  const str = createString();
  const buffer = Buffer.from(str, "utf8");

  let uploadId;

  try {
    const multipartUpload = await s3Client.send(
      new CreateMultipartUploadCommand({
        Bucket: bucketName,
        Key: key,
      }),
    );

    uploadId = multipartUpload.UploadId;

    const uploadPromises = [];
    // Multipart uploads require a minimum size of 5 MB per part.
    const partSize = Math.ceil(buffer.length / 5);

    // Upload each part.
    for (let i = 0; i < 5; i++) {
      const start = i * partSize;
      const end = start + partSize;
      uploadPromises.push(
        s3Client
          .send(
            new UploadPartCommand({
              Bucket: bucketName,
```

```
            Key: key,
            UploadId: uploadId,
            Body: buffer.subarray(start, end),
            PartNumber: i + 1,
          }),
        )
        .then((d) => {
          console.log("Part", i + 1, "uploaded");
          return d;
        }),
    );
  }

  const uploadResults = await Promise.all(uploadPromises);

  return await s3Client.send(
    new CompleteMultipartUploadCommand({
      Bucket: bucketName,
      Key: key,
      UploadId: uploadId,
      MultipartUpload: {
        Parts: uploadResults.map(({ ETag }, i) => ({
          ETag,
          PartNumber: i + 1,
        })),
      },
    }),
  );

  // Verify the output by downloading the file from the Amazon Simple Storage
Service (Amazon S3) console.
  // Because the output is a 25 MB string, text editors might struggle to open the
file.
 } catch (err) {
  console.error(err);

  if (uploadId) {
    const abortCommand = new AbortMultipartUploadCommand({
      Bucket: bucketName,
      Key: key,
      UploadId: uploadId,
    });

    await s3Client.send(abortCommand);
```

```
    }
  }
};
```

下载大文件。

```
import { GetObjectCommand, S3Client } from "@aws-sdk/client-s3";
import { createWriteStream } from "fs";

const s3Client = new S3Client({});
const oneMB = 1024 * 1024;

export const getObjectRange = ({ bucket, key, start, end }) => {
  const command = new GetObjectCommand({
    Bucket: bucket,
    Key: key,
    Range: `bytes=${start}-${end}`,
  });

  return s3Client.send(command);
};

export const getRangeAndLength = (contentRange) => {
  const [range, length] = contentRange.split("/");
  const [start, end] = range.split("-");
  return {
    start: parseInt(start),
    end: parseInt(end),
    length: parseInt(length),
  };
};

export const isComplete = ({ end, length }) => end === length - 1;

// When downloading a large file, you might want to break it down into
// smaller pieces. Amazon S3 accepts a Range header to specify the start
// and end of the byte range to be downloaded.
const downloadInChunks = async ({ bucket, key }) => {
  const writeStream = createWriteStream(
    fileURLToPath(new URL(`./${key}`, import.meta.url))
  ).on("error", (err) => console.error(err));
```

```
  let rangeAndLength = { start: -1, end: -1, length: -1 };

  while (!isComplete(rangeAndLength)) {
    const { end } = rangeAndLength;
    const nextRange = { start: end + 1, end: end + oneMB };

    console.log(`Downloading bytes ${nextRange.start} to ${nextRange.end}`);

    const { ContentRange, Body } = await getObjectRange({
      bucket,
      key,
      ...nextRange,
    });

    writeStream.write(await Body.transformToByteArray());
    rangeAndLength = getRangeAndLength(ContentRange);
  }
};

export const main = async () => {
  await downloadInChunks({
    bucket: "my-cool-bucket",
    key: "my-cool-object.txt",
  });
};
```

使用 JavaScript (v3) 软件开发工具包的 S3 Glacier 示例

以下代码示例向您展示了如何使用带有 S3 Glacier 的 Amazon SDK for JavaScript (v3) 来执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景 是展示如何通过在同一服务中调用多个函数来完成特定任务的代码示例。

每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关如何在上下文中设置和运行代码的说明。

主题

- 操作

操作

创建文件库

以下代码示例展示了如何创建 Amazon S3 Glacier 保管库。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。查找完整示例，了解如何在 Amazon 代码示例存储库中进行设置和运行。

创建客户端。

```
const { GlacierClient } = require("@aws-sdk/client-glacier");
// Set the AWS Region.
const REGION = "REGION";
//Set the Redshift Service Object
const glacierClient = new GlacierClient({ region: REGION });
export { glacierClient };
```

创建文件库。

```
// Load the SDK for JavaScript
import { CreateVaultCommand } from "@aws-sdk/client-glacier";
import { glacierClient } from "./libs/glacierClient.js";

// Set the parameters
const vaultname = "VAULT_NAME"; // VAULT_NAME
const params = { vaultName: vaultname };

const run = async () => {
  try {
    const data = await glacierClient.send(new CreateVaultCommand(params));
    console.log("Success, vault created!");
    return data; // For unit tests.
  } catch (err) {
    console.log("Error");
```

```
  }
};
run();
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考CreateVault中的。

适用于 JavaScript (v2) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
// Load the SDK for JavaScript
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create a new service object
var glacier = new AWS.Glacier({ apiVersion: "2012-06-01" });
// Call Glacier to create the vault
glacier.createVault({ vaultName: "YOUR_VAULT_NAME" }, function (err) {
  if (!err) {
    console.log("Created vault!");
  }
});
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考CreateVault中的。

将存档上传到文件库

以下代码示例展示了如何将存档上传至 Amazon S3 Glacier 保管库。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。查找完整示例，了解如何在 Amazon 代码示例存储库中进行设置和运行。

创建客户端。

```
const { GlacierClient } = require("@aws-sdk/client-glacier");
// Set the AWS Region.
const REGION = "REGION";
//Set the Redshift Service Object
const glacierClient = new GlacierClient({ region: REGION });
export { glacierClient };
```

上传档案。

```
// Load the SDK for JavaScript
import { UploadArchiveCommand } from "@aws-sdk/client-glacier";
import { glacierClient } from "./libs/glacierClient.js";

// Set the parameters
const vaultname = "VAULT_NAME"; // VAULT_NAME

// Create a new service object and buffer
const buffer = new Buffer.alloc(2.5 * 1024 * 1024); // 2.5MB buffer
const params = { vaultName: vaultname, body: buffer };

const run = async () => {
  try {
    const data = await glacierClient.send(new UploadArchiveCommand(params));
    console.log("Archive ID", data.archiveId);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error uploading archive!", err);
  }
};
run();
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考UploadArchive中的。

适用于 JavaScript (v2) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
// Load the SDK for JavaScript
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create a new service object and buffer
var glacier = new AWS.Glacier({ apiVersion: "2012-06-01" });
buffer = Buffer.alloc(2.5 * 1024 * 1024); // 2.5MB buffer

var params = { vaultName: "YOUR_VAULT_NAME", body: buffer };
// Call Glacier to upload the archive.
glacier.uploadArchive(params, function (err, data) {
  if (err) {
    console.log("Error uploading archive!", err);
  } else {
    console.log("Archive ID", data.archiveId);
  }
});
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考UploadArchive中的。

SageMaker 使用适用于 JavaScript (v3) 的 SDK 的示例

以下代码示例向您展示了如何通过使用 Amazon SDK for JavaScript (v3) 来执行操作和实现常见场景 SageMaker。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景 是展示如何通过在同一服务中调用多个函数来完成特定任务的代码示例。

每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关如何在上下文中设置和运行代码的说明。

开始使用

你好 SageMaker

以下代码示例展示了如何开始使用 SageMaker。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import {
  SageMakerClient,
  ListNotebookInstancesCommand,
} from "@aws-sdk/client-sagemaker";

const client = new SageMakerClient({
  region: "us-west-2",
});

export const helloSagemaker = async () => {
  const command = new ListNotebookInstancesCommand({ MaxResults: 5 });

  const response = await client.send(command);
  console.log(
    "Hello Amazon SageMaker! Let's list some of your notebook instances:",
  );

  const instances = response.NotebookInstances || [];

  if (instances.length === 0) {
    console.log(
      "• No notebook instances found. Try creating one in the AWS Management Console
 or with the CreateNotebookInstanceCommand.",
```

```
      );
    } else {
      console.log(
        instances
          .map(
            (i) =>
              `• Instance: ${i.NotebookInstanceName}\n  Arn:${
                i.NotebookInstanceArn
              } \n  Creation Date: ${i.CreationTime.toISOString()}`,
          )
          .join("\n"),
      );
    }

    return response;
  };
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考ListNotebookInstances中的。

主题

- 操作
- 场景

操作

创建管道

以下代码示例显示了如何在中创建或更新管道 SageMaker。

适用于 JavaScript (v3) 的软件开发工具包

Note

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

使用本地提供的 JSON 定义创建 SageMaker 管道的函数。

```
/**
 * Create the Amazon SageMaker pipeline using a JSON pipeline definition. The
 definition
 * can also be provided as an Amazon S3 object using PipelineDefinitionS3Location.
 * @param {{roleArn: string, name: string, sagemakerClient: import('@aws-sdk/client-
sagemaker').SageMakerClient}} props
 */
export async function createSagemakerPipeline({
  // Assumes an AWS IAM role has been created for this pipeline.
  roleArn,
  name,
  // Assumes an AWS Lambda function has been created for this pipeline.
  functionArn,
  sagemakerClient,
}) {
  const pipelineDefinition = readFileSync(
    // dirnameFromMetaUrl is a local utility function. You can find its
 implementation
    // on GitHub.
    `${dirnameFromMetaUrl(
      import.meta.url,
    )}../../../../../workflows/sagemaker_pipelines/resources/
GeoSpatialPipeline.json`,
  )
    .toString()
    .replace(/\*FUNCTION_ARN\*/g, functionArn);

  const { PipelineArn } = await sagemakerClient.send(
    new CreatePipelineCommand({
      PipelineName: name,
      PipelineDefinition: pipelineDefinition,
      RoleArn: roleArn,
    }),
  );

  return {
    arn: PipelineArn,
    cleanUp: async () => {
      await sagemakerClient.send(
        new DeletePipelineCommand({ PipelineName: name }),
      );
    },
  };
```

```
}
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考中的以下主题。
  - CreatePipeline
  - UpdatePipeline

删除管道

以下代码示例说明如何删除中的管道 SageMaker。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

删除 SageMaker 管道的语法。这段代码是更大函数的一部分。有关更多上下文，请参阅 “创建管道” 或 GitHub 存储库。

```
    await sagemakerClient.send(
      new DeletePipelineCommand({ PipelineName: name }),
    );
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DeletePipeline中的。

描述管道执行

以下代码示例说明如何描述中的管道执行 SageMaker。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

等待 SageMaker 管道执行成功、失败或停止。

```
/**
 * Poll the executing pipeline until the status is 'SUCCEEDED', 'STOPPED', or
 'FAILED'.
 * @param {{ arn: string, sagemakerClient: import('@aws-sdk/client-
sagemaker').SageMakerClient}} props
 */
export async function waitForPipelineComplete({ arn, sagemakerClient }) {
  const command = new DescribePipelineExecutionCommand({
    PipelineExecutionArn: arn,
  });

  let complete = false;
  let intervalInSeconds = 15;
  const COMPLETION_STATUSES = [
    PipelineExecutionStatus.FAILED,
    PipelineExecutionStatus.STOPPED,
    PipelineExecutionStatus.SUCCEEDED,
  ];

  do {
    const { PipelineExecutionStatus: status, FailureReason } =
      await sagemakerClient.send(command);

    complete = COMPLETION_STATUSES.includes(status);

    if (!complete) {
      console.log(
        `Pipeline is ${status}. Waiting ${intervalInSeconds} seconds before checking
 again.`,
      );
      await wait(intervalInSeconds);
    } else if (status === PipelineExecutionStatus.FAILED) {
      throw new Error(`Pipeline failed because: ${FailureReason}`);
    } else if (status === PipelineExecutionStatus.STOPPED) {
      throw new Error(`Pipeline was forcefully stopped.`);
    } else {
      console.log(`Pipeline execution ${status}.`);
    }
  } while (!complete);
}
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DescribePipelineExecution中的。

执行管道

以下代码示例显示了如何在中启动管道执行 SageMaker。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

开始 SageMaker 管道执行。

```
/**
 * Start the execution of the Amazon SageMaker pipeline. Parameters that are
 * passed in are used in the AWS Lambda function.
 * @param {{
 *   name: string,
 *   sagemakerClient: import('@aws-sdk/client-sagemaker').SageMakerClient,
 *   roleArn: string,
 *   queueUrl: string,
 *   s3InputBucketName: string,
 * }} props
 */
export async function startPipelineExecution({
  sagemakerClient,
  name,
  bucketName,
  roleArn,
  queueUrl,
}) {
  /**
   * The Vector Enrichment Job requests CSV data. This configuration points to a CSV
   * file in an Amazon S3 bucket.
   * @type {import("@aws-sdk/client-sagemaker-
geospatial").VectorEnrichmentJobInputConfig}
   */
  const inputConfig = {
```

```
    DataSourceConfig: {
      S3Data: {
        S3Uri: `s3://${bucketName}/input/sample_data.csv`,
      },
    },
    DocumentType: VectorEnrichmentJobDocumentType.CSV,
  };

  /**
   * The Vector Enrichment Job adds additional data to the source CSV. This
 configuration points
   * to an Amazon S3 prefix where the output will be stored.
   * @type {import("@aws-sdk/client-sagemaker-
geospatial").ExportVectorEnrichmentJobOutputConfig}
   */
  const outputConfig = {
    S3Data: {
      S3Uri: `s3://${bucketName}/output/`,
    },
  };

  /**
   * This job will be a Reverse Geocoding Vector Enrichment Job. Reverse Geocoding
 requires
   * latitude and longitude values.
   * @type {import("@aws-sdk/client-sagemaker-
geospatial").VectorEnrichmentJobConfig}
   */
  const jobConfig = {
    ReverseGeocodingConfig: {
      XAttributeName: "Longitude",
      YAttributeName: "Latitude",
    },
  };

  const { PipelineExecutionArn } = await sagemakerClient.send(
    new StartPipelineExecutionCommand({
      PipelineName: name,
      PipelineExecutionDisplayName: `${name}-example-execution`,
      PipelineParameters: [
        { Name: "parameter_execution_role", Value: roleArn },
        { Name: "parameter_queue_url", Value: queueUrl },
        {
          Name: "parameter_vej_input_config",
```

```
          Value: JSON.stringify(inputConfig),
        },
        {
          Name: "parameter_vej_export_config",
          Value: JSON.stringify(outputConfig),
        },
        {
          Name: "parameter_step_1_vej_config",
          Value: JSON.stringify(jobConfig),
        },
      ],
    }),
  );

  return {
    arn: PipelineExecutionArn,
  };
}
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考StartPipelineExecution中的。

场景

开始使用地理空间作业和管道

以下代码示例演示了操作流程：

- 为管道设置资源。
- 设置用于执行地理空间作业的管道。
- 启动管道执行。
- 监控执行的状态。
- 查看管道的输出。
- 清理资源。

有关更多信息，请参阅在 Community.aw Amazon s 上使用软件开发工具包创建和运行 SageMaker 管道。

适用于 JavaScript (v3) 的软件开发工具包

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

以下文件摘录包含使用 SageMaker 客户端管理管道的函数。

```
import { readFileSync } from "fs";

import {
  CreateRoleCommand,
  DeleteRoleCommand,
  CreatePolicyCommand,
  DeletePolicyCommand,
  AttachRolePolicyCommand,
  DetachRolePolicyCommand,
} from "@aws-sdk/client-iam";

import {
  PublishLayerVersionCommand,
  DeleteLayerVersionCommand,
  CreateFunctionCommand,
  Runtime,
  DeleteFunctionCommand,
  CreateEventSourceMappingCommand,
  DeleteEventSourceMappingCommand,
} from "@aws-sdk/client-lambda";

import {
  PutObjectCommand,
  CreateBucketCommand,
  DeleteBucketCommand,
  paginateListObjectsV2,
  DeleteObjectCommand,
  GetObjectCommand,
  ListObjectsV2Command,
} from "@aws-sdk/client-s3";

import {
  CreatePipelineCommand,
```

```
  DeletePipelineCommand,
  DescribePipelineExecutionCommand,
  PipelineExecutionStatus,
  StartPipelineExecutionCommand,
} from "@aws-sdk/client-sagemaker";

import { VectorEnrichmentJobDocumentType } from "@aws-sdk/client-sagemaker-
geospatial";

import {
  CreateQueueCommand,
  DeleteQueueCommand,
  GetQueueAttributesCommand,
} from "@aws-sdk/client-sqs";

import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";
import { retry, wait } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

/**
 * Create the AWS IAM role that will be assumed by AWS Lambda.
 * @param {{ name: string, iamClient: import('@aws-sdk/client-iam').IAMClient }}
 props
 */
export async function createLambdaExecutionRole({ name, iamClient }) {
  const { Role } = await iamClient.send(
    new CreateRoleCommand({
      RoleName: name,
      AssumeRolePolicyDocument: JSON.stringify({
        Version: "2012-10-17",
        Statement: [
          {
            Effect: "Allow",
            Action: ["sts:AssumeRole"],
            Principal: { Service: ["lambda.amazonaws.com"] },
          },
        ],
      }),
    }),
  );

  return {
    arn: Role.Arn,
    cleanUp: async () => {
      await iamClient.send(new DeleteRoleCommand({ RoleName: name }));
```

```
    },
  };
}

/**
 * Create an AWS IAM policy that will be attached to the AWS IAM role assumed by the
 AWS Lambda function.
 * The policy grants permission to work with Amazon SQS, Amazon CloudWatch, and
 Amazon SageMaker.
 * @param {{name: string, iamClient: import('@aws-sdk/client-iam').IAMClient,
 pipelineExecutionRoleArn: string}} props
 */
export async function createLambdaExecutionPolicy({
  name,
  iamClient,
  pipelineExecutionRoleArn,
}) {
  const policy = {
    Version: "2012-10-17",
    Statement: [
      {
        Effect: "Allow",
        Action: [
          "sqs:ReceiveMessage",
          "sqs:DeleteMessage",
          "sqs:GetQueueAttributes",
          "logs:CreateLogGroup",
          "logs:CreateLogStream",
          "logs:PutLogEvents",
          "sagemaker-geospatial:StartVectorEnrichmentJob",
          "sagemaker-geospatial:GetVectorEnrichmentJob",
          "sagemaker:SendPipelineExecutionStepFailure",
          "sagemaker:SendPipelineExecutionStepSuccess",
          "sagemaker-geospatial:ExportVectorEnrichmentJob",
        ],
        Resource: "*",
      },
      {
        Effect: "Allow",
        // The AWS Lambda function needs permission to pass the pipeline execution
 role to
        // the StartVectorEnrichmentCommand. This restriction prevents an AWS Lambda
 function
        // from elevating privileges. For more information, see:
```

```
        // https://docs.aws.amazon.com/IAM/latest/UserGuide/
id_roles_use_passrole.html
        Action: ["iam:PassRole"],
        Resource: `${pipelineExecutionRoleArn}`,
        Condition: {
          StringEquals: {
            "iam:PassedToService": [
              "sagemaker.amazonaws.com",
              "sagemaker-geospatial.amazonaws.com",
            ],
          },
        },
      },
    ],
  };

  const createPolicyCommand = new CreatePolicyCommand({
    PolicyDocument: JSON.stringify(policy),
    PolicyName: name,
  });

  const { Policy } = await iamClient.send(createPolicyCommand);
  return {
    arn: Policy.Arn,
    policy,
    cleanUp: async () => {
      await iamClient.send(new DeletePolicyCommand({ PolicyArn: Policy.Arn }));
    },
  };
}

/**
 * Attach an AWS IAM policy to an AWS IAM role.
 * @param {{roleName: string, policyArn: string, iamClient: import('@aws-sdk/client-
iam').IAMClient}} props
 */
export async function attachPolicy({ roleName, policyArn, iamClient }) {
  const attachPolicyCommand = new AttachRolePolicyCommand({
    RoleName: roleName,
    PolicyArn: policyArn,
  });

  await iamClient.send(attachPolicyCommand);
  return {
```

```
    cleanUp: async () => {
      await iamClient.send(
        new DetachRolePolicyCommand({
          RoleName: roleName,
          PolicyArn: policyArn,
        }),
      );
    },
  };
}

/**
 * Create an AWS Lambda layer that contains the Amazon SageMaker and Amazon
 SageMaker Geospatial clients
 * in the runtime. The default runtime supports v3.188.0 of the JavaScript SDK. The
 Amazon SageMaker
 * Geospatial client wasn't introduced until v3.221.0.
 * @param {{ name: string, lambdaClient: import('@aws-sdk/client-
lambda').LambdaClient }} props
 */
export async function createLambdaLayer({ name, lambdaClient }) {
  const layerPath = `${dirnameFromMetaUrl(import.meta.url)}lambda/nodejs.zip`;
  const { LayerVersionArn, Version } = await lambdaClient.send(
    new PublishLayerVersionCommand({
      LayerName: name,
      Content: {
        ZipFile: Uint8Array.from(readFileSync(layerPath)),
      },
    }),
  );

  return {
    versionArn: LayerVersionArn,
    version: Version,
    cleanUp: async () => {
      await lambdaClient.send(
        new DeleteLayerVersionCommand({
          LayerName: name,
          VersionNumber: Version,
        }),
      );
    },
  };
}
```

```
/**
 * Deploy the AWS Lambda function that will be used to respond to Amazon SageMaker
 pipeline
 * execution steps.
 * @param {{roleArn: string, name: string, lambdaClient: import('@aws-sdk/client-
lambda').LambdaClient, layerVersionArn: string}} props
 */
export async function createLambdaFunction({
  name,
  roleArn,
  lambdaClient,
  layerVersionArn,
}) {
  const lambdaPath = `${dirnameFromMetaUrl(
    import.meta.url,
  )}lambda/dist/index.mjs.zip`;

  const command = new CreateFunctionCommand({
    Code: {
      ZipFile: Uint8Array.from(readFileSync(lambdaPath)),
    },
    Runtime: Runtime.nodejs18x,
    Handler: "index.handler",
    Layers: [layerVersionArn],
    FunctionName: name,
    Role: roleArn,
  });

  // Function creation fails if the Role is not ready. This retries
  // function creation until it succeeds or it times out.
  const { FunctionArn } = await retry(
    { intervalInMs: 1000, maxRetries: 60 },
    () => lambdaClient.send(command),
  );

  return {
    arn: FunctionArn,
    cleanUp: async () => {
      await lambdaClient.send(
        new DeleteFunctionCommand({ FunctionName: name }),
      );
    },
  };
```

```
}

/**
 * This uploads some sample coordinate data to an Amazon S3 bucket.
 * The Amazon SageMaker Geospatial vector enrichment job will take the simple Lat/
Long
 * coordinates in this file and augment them with more detailed location data.
 * @param {{bucketName: string, s3Client: import('@aws-sdk/client-s3').S3Client}}
 props
 */
export async function uploadCSVDataToS3({ bucketName, s3Client }) {
  const s3Path = `${dirnameFromMetaUrl(
    import.meta.url,
  )}../../../../../workflows/sagemaker_pipelines/resources/latlongtest.csv`;

  await s3Client.send(
    new PutObjectCommand({
      Bucket: bucketName,
      Key: "input/sample_data.csv",
      Body: readFileSync(s3Path),
    }),
  );
}

/**
 * Create the AWS IAM role that will be assumed by the Amazon SageMaker pipeline.
 * @param {{name: string, iamClient: import('@aws-sdk/client-iam').IAMClient}} props
 */
export async function createSagemakerRole({ name, iamClient }) {
  const command = new CreateRoleCommand({
    RoleName: name,
    AssumeRolePolicyDocument: JSON.stringify({
      Version: "2012-10-17",
      Statement: [
        {
          Effect: "Allow",
          Action: ["sts:AssumeRole"],
          Principal: {
            Service: [
              "sagemaker.amazonaws.com",
              "sagemaker-geospatial.amazonaws.com",
            ],
          },
        },
```

```
      ],
    }),
  });

  const { Role } = await iamClient.send(command);
  // Wait for the role to be ready.
  await wait(10);

  return {
    arn: Role.Arn,
    cleanUp: async () => {
      await iamClient.send(new DeleteRoleCommand({ RoleName: name }));
    },
  };
}

/**
 * Create the Amazon SageMaker execution policy. This policy grants permission to
 * invoke the AWS Lambda function, read/write to the Amazon S3 bucket, and send
 messages to
 * the Amazon SQS queue.
 * @param {{ name: string, sqsQueueArn: string, lambdaArn: string, iamClient:
 import('@aws-sdk/client-iam').IAMClient, s3BucketName: string}} props
 */
export async function createSagemakerExecutionPolicy({
  sqsQueueArn,
  lambdaArn,
  iamClient,
  name,
  s3BucketName,
}) {
  const policy = {
    Version: "2012-10-17",
    Statement: [
      {
        Effect: "Allow",
        Action: ["lambda:InvokeFunction"],
        Resource: lambdaArn,
      },
      {
        Effect: "Allow",
        Action: ["s3:*"],
        Resource: [
          `arn:aws:s3:::${s3BucketName}`,
```

```
          `arn:aws:s3:::${s3BucketName}/*`,
        ],
      },
      {
        Effect: "Allow",
        Action: ["sqs:SendMessage"],
        Resource: sqsQueueArn,
      },
    ],
  };

  const createPolicyCommand = new CreatePolicyCommand({
    PolicyDocument: JSON.stringify(policy),
    PolicyName: name,
  });

  const { Policy } = await iamClient.send(createPolicyCommand);
  return {
    arn: Policy.Arn,
    policy,
    cleanUp: async () => {
      await iamClient.send(new DeletePolicyCommand({ PolicyArn: Policy.Arn }));
    },
  };
}

/**
 * Create the Amazon SageMaker pipeline using a JSON pipeline definition. The
 definition
 * can also be provided as an Amazon S3 object using PipelineDefinitionS3Location.
 * @param {{roleArn: string, name: string, sagemakerClient: import('@aws-sdk/client-
sagemaker').SageMakerClient}} props
 */
export async function createSagemakerPipeline({
  // Assumes an AWS IAM role has been created for this pipeline.
  roleArn,
  name,
  // Assumes an AWS Lambda function has been created for this pipeline.
  functionArn,
  sagemakerClient,
}) {
  const pipelineDefinition = readFileSync(
    // dirnameFromMetaUrl is a local utility function. You can find its
 implementation
```

```
    // on GitHub.
    `${dirnameFromMetaUrl(
      import.meta.url,
    )}../../../../../workflows/sagemaker_pipelines/resources/
GeoSpatialPipeline.json`,
  )
    .toString()
    .replace(/\*FUNCTION_ARN\*/g, functionArn);

  const { PipelineArn } = await sagemakerClient.send(
    new CreatePipelineCommand({
      PipelineName: name,
      PipelineDefinition: pipelineDefinition,
      RoleArn: roleArn,
    }),
  );

  return {
    arn: PipelineArn,
    cleanUp: async () => {
      await sagemakerClient.send(
        new DeletePipelineCommand({ PipelineName: name }),
      );
    },
  };
}

/**
 * Create an Amazon SQS queue. The Amazon SageMaker pipeline will send messages
 * to this queue that are then processed by the AWS Lambda function.
 * @param {{name: string, sqsClient: import('@aws-sdk/client-sqs').SQSClient}} props
 */
export async function createSQSQueue({ name, sqsClient }) {
  const { QueueUrl } = await sqsClient.send(
    new CreateQueueCommand({
      QueueName: name,
      Attributes: {
        DelaySeconds: "5",
        ReceiveMessageWaitTimeSeconds: "5",
        VisibilityTimeout: "300",
      },
    }),
  );
```

```
  const { Attributes } = await sqsClient.send(
    new GetQueueAttributesCommand({
      QueueUrl,
      AttributeNames: ["QueueArn"],
    }),
  );

  return {
    queueUrl: QueueUrl,
    queueArn: Attributes.QueueArn,
    cleanUp: async () => {
      await sqsClient.send(new DeleteQueueCommand({ QueueUrl }));
    },
  };
}

/**
 * Configure the AWS Lambda function to long poll for messages from the Amazon SQS
 * queue.
 * @param {{lambdaName: string, queueArn: string, lambdaClient: import('@aws-sdk/
client-lambda').LambdaClient, sqsClient: import('@aws-sdk/client-sqs').SQSClient}}
 props
 */
export async function configureLambdaSQSEventSource({
  lambdaName,
  queueArn,
  lambdaClient,
}) {
  const { UUID } = await lambdaClient.send(
    new CreateEventSourceMappingCommand({
      EventSourceArn: queueArn,
      FunctionName: lambdaName,
    }),
  );

  return {
    cleanUp: async () => {
      await lambdaClient.send(
        new DeleteEventSourceMappingCommand({
          UUID,
        }),
      );
    },
  };
```

```
}

/**
 * Create an Amazon S3 bucket that will store the simple coordinate file as input
 * and the output of the Amazon SageMaker Geospatial vector enrichment job.
 * @param {{s3Client: import('@aws-sdk/client-s3').S3Client, name: string}} props
 */
export async function createS3Bucket({ name, s3Client }) {
  await s3Client.send(new CreateBucketCommand({ Bucket: name }));

  return {
    cleanUp: async () => {
      const paginator = paginateListObjectsV2(
        { client: s3Client },
        { Bucket: name },
      );
      for await (const page of paginator) {
        const objects = page.Contents;
        if (objects) {
          for (const object of objects) {
            await s3Client.send(
              new DeleteObjectCommand({ Bucket: name, Key: object.Key }),
            );
          }
        }
      }
      await s3Client.send(new DeleteBucketCommand({ Bucket: name }));
    },
  };
}

/**
 * Start the execution of the Amazon SageMaker pipeline. Parameters that are
 * passed in are used in the AWS Lambda function.
 * @param {{
 *   name: string,
 *   sagemakerClient: import('@aws-sdk/client-sagemaker').SageMakerClient,
 *   roleArn: string,
 *   queueUrl: string,
 *   s3InputBucketName: string,
 * }} props
 */
export async function startPipelineExecution({
  sagemakerClient,
```

```
  name,
  bucketName,
  roleArn,
  queueUrl,
}) {
  /**
   * The Vector Enrichment Job requests CSV data. This configuration points to a CSV
   * file in an Amazon S3 bucket.
   * @type {import("@aws-sdk/client-sagemaker-
geospatial").VectorEnrichmentJobInputConfig}
   */
  const inputConfig = {
    DataSourceConfig: {
      S3Data: {
        S3Uri: `s3://${bucketName}/input/sample_data.csv`,
      },
    },
    DocumentType: VectorEnrichmentJobDocumentType.CSV,
  };

  /**
   * The Vector Enrichment Job adds additional data to the source CSV. This
 configuration points
   * to an Amazon S3 prefix where the output will be stored.
   * @type {import("@aws-sdk/client-sagemaker-
geospatial").ExportVectorEnrichmentJobOutputConfig}
   */
  const outputConfig = {
    S3Data: {
      S3Uri: `s3://${bucketName}/output/`,
    },
  };

  /**
   * This job will be a Reverse Geocoding Vector Enrichment Job. Reverse Geocoding
 requires
   * latitude and longitude values.
   * @type {import("@aws-sdk/client-sagemaker-
geospatial").VectorEnrichmentJobConfig}
   */
  const jobConfig = {
    ReverseGeocodingConfig: {
      XAttributeName: "Longitude",
      YAttributeName: "Latitude",
```

```
    },
  };

  const { PipelineExecutionArn } = await sagemakerClient.send(
    new StartPipelineExecutionCommand({
      PipelineName: name,
      PipelineExecutionDisplayName: `${name}-example-execution`,
      PipelineParameters: [
        { Name: "parameter_execution_role", Value: roleArn },
        { Name: "parameter_queue_url", Value: queueUrl },
        {
          Name: "parameter_vej_input_config",
          Value: JSON.stringify(inputConfig),
        },
        {
          Name: "parameter_vej_export_config",
          Value: JSON.stringify(outputConfig),
        },
        {
          Name: "parameter_step_1_vej_config",
          Value: JSON.stringify(jobConfig),
        },
      ],
    }),
  );

  return {
    arn: PipelineExecutionArn,
  };
}

/**
 * Poll the executing pipeline until the status is 'SUCCEEDED', 'STOPPED', or
 'FAILED'.
 * @param {{ arn: string, sagemakerClient: import('@aws-sdk/client-
sagemaker').SageMakerClient}} props
 */
export async function waitForPipelineComplete({ arn, sagemakerClient }) {
  const command = new DescribePipelineExecutionCommand({
    PipelineExecutionArn: arn,
  });

  let complete = false;
  let intervalInSeconds = 15;
```

```
  const COMPLETION_STATUSES = [
    PipelineExecutionStatus.FAILED,
    PipelineExecutionStatus.STOPPED,
    PipelineExecutionStatus.SUCCEEDED,
  ];

  do {
    const { PipelineExecutionStatus: status, FailureReason } =
      await sagemakerClient.send(command);

    complete = COMPLETION_STATUSES.includes(status);

    if (!complete) {
      console.log(
        `Pipeline is ${status}. Waiting ${intervalInSeconds} seconds before checking
 again.`,
      );
      await wait(intervalInSeconds);
    } else if (status === PipelineExecutionStatus.FAILED) {
      throw new Error(`Pipeline failed because: ${FailureReason}`);
    } else if (status === PipelineExecutionStatus.STOPPED) {
      throw new Error(`Pipeline was forcefully stopped.`);
    } else {
      console.log(`Pipeline execution ${status}.`);
    }
  } while (!complete);
}

/**
 * Return the string value of an Amazon S3 object.
 * @param {{ bucket: string, key: string, s3Client: import('@aws-sdk/client-
s3').S3Client}} param0
 */
export async function getObject({ bucket, s3Client }) {
  const prefix = "output/";
  const { Contents } = await s3Client.send(
    new ListObjectsV2Command({ MaxKeys: 1, Bucket: bucket, Prefix: prefix }),
  );

  if (!Contents.length) {
    throw new Error("No objects found in bucket.");
  }

  // Find the CSV file.
```

```
  const outputObject = Contents.find((obj) => obj.Key.endsWith(".csv"));

  if (!outputObject) {
    throw new Error(`No CSV file found in bucket with the prefix "${prefix}".`);
  }

  const { Body } = await s3Client.send(
    new GetObjectCommand({
      Bucket: bucket,
      Key: outputObject.Key,
    }),
  );

  return Body.transformToString();
}
```

此函数摘自一个文件，该文件使用前面的库函数来设置 SageMaker 管道、执行管道并删除所有已创建的资源。

```
import { retry, wait } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";
import {
  attachPolicy,
  configureLambdaSQSEventSource,
  createLambdaExecutionPolicy,
  createLambdaExecutionRole,
  createLambdaFunction,
  createLambdaLayer,
  createS3Bucket,
  createSQSQueue,
  createSagemakerExecutionPolicy,
  createSagemakerPipeline,
  createSagemakerRole,
  getObject,
  startPipelineExecution,
  uploadCSVDataToS3,
  waitForPipelineComplete,
} from "./lib.js";
import { MESSAGES } from "./messages.js";

export class SageMakerPipelinesWkflw {
  names = {
    LAMBDA_EXECUTION_ROLE: "sagemaker-wkflw-lambda-execution-role",
```

```
    LAMBDA_EXECUTION_ROLE_POLICY:
      "sagemaker-wkflw-lambda-execution-role-policy",
    LAMBDA_FUNCTION: "sagemaker-wkflw-lambda-function",
    LAMBDA_LAYER: "sagemaker-wkflw-lambda-layer",
    SAGE_MAKER_EXECUTION_ROLE: "sagemaker-wkflw-pipeline-execution-role",
    SAGE_MAKER_EXECUTION_ROLE_POLICY:
      "sagemaker-wkflw-pipeline-execution-role-policy",
    SAGE_MAKER_PIPELINE: "sagemaker-wkflw-pipeline",
    SQS_QUEUE: "sagemaker-wkflw-sqs-queue",
    S3_BUCKET: `sagemaker-wkflw-s3-bucket-${Date.now()}`,
  };

  cleanUpFunctions = [];

  /**
   * @param {import("@aws-doc-sdk-examples/lib/prompter.js").Prompter} prompter
   * @param {import("@aws-doc-sdk-examples/lib/logger.js").Logger} logger
   * @param {{ IAM: import("@aws-sdk/client-iam").IAMClient, Lambda: import("@aws-
sdk/client-lambda").LambdaClient, SageMaker: import("@aws-sdk/client-
sagemaker").SageMakerClient, S3: import("@aws-sdk/client-s3").S3Client, SQS:
 import("@aws-sdk/client-sqs").SQSClient }} clients
   */
  constructor(prompter, logger, clients) {
    this.prompter = prompter;
    this.logger = logger;
    this.clients = clients;
  }

  async run() {
    try {
      await this.startWorkflow();
    } catch (err) {
      console.error(err);
      throw err;
    } finally {
      // Run all of the clean up functions. If any fail, we log the error and
 continue.
      // This ensures all clean up functions are run.
      this.logger.logSeparator();
      const doCleanUp = await this.prompter.confirm({
        message: "Clean up resources?",
      });
      if (doCleanUp) {
        for (let i = this.cleanUpFunctions.length - 1; i >= 0; i--) {
```

```
          await retry(
            { intervalInMs: 1000, maxRetries: 60, swallowError: true },
            this.cleanUpFunctions[i],
          );
        }
      }
    }
  }

  async startWorkflow() {
    this.logger.logSeparator(MESSAGES.greetingHeader);
    await this.logger.log(MESSAGES.greeting);

    this.logger.logSeparator();
    await this.logger.log(
      MESSAGES.creatingRole.replace(
        "${ROLE_NAME}",
        this.names.LAMBDA_EXECUTION_ROLE,
      ),
    );

    // Create an IAM role that will be assumed by the AWS Lambda function. This
 function
    // is triggered by Amazon SQS messages and calls SageMaker and SageMaker
 GeoSpatial actions.
    const { arn: lambdaExecutionRoleArn, cleanUp: lambdaExecutionRoleCleanUp } =
      await createLambdaExecutionRole({
        name: this.names.LAMBDA_EXECUTION_ROLE,
        iamClient: this.clients.IAM,
      });
    // Add a clean up step to a stack for every resource created.
    this.cleanUpFunctions.push(lambdaExecutionRoleCleanUp);

    await this.logger.log(
      MESSAGES.roleCreated.replace(
        "${ROLE_NAME}",
        this.names.LAMBDA_EXECUTION_ROLE,
      ),
    );

    this.logger.logSeparator();

    await this.logger.log(
      MESSAGES.creatingRole.replace(
```

```
        "${ROLE_NAME}",
        this.names.SAGE_MAKER_EXECUTION_ROLE,
      ),
    );

    // Create an IAM role that will be assumed by the SageMaker pipeline. The
 pipeline
    // sends messages to an Amazon SQS queue and puts/retrieves Amazon S3 objects.
    const {
      arn: pipelineExecutionRoleArn,
      cleanUp: pipelineExecutionRoleCleanUp,
    } = await createSagemakerRole({
      iamClient: this.clients.IAM,
      name: this.names.SAGE_MAKER_EXECUTION_ROLE,
    });
    this.cleanUpFunctions.push(pipelineExecutionRoleCleanUp);

    await this.logger.log(
      MESSAGES.roleCreated.replace(
        "${ROLE_NAME}",
        this.names.SAGE_MAKER_EXECUTION_ROLE,
      ),
    );

    this.logger.logSeparator();

    // Create an IAM policy that allows the AWS Lambda function to invoke SageMaker
 APIs.
    const {
      arn: lambdaExecutionPolicyArn,
      policy: lambdaPolicy,
      cleanUp: lambdaExecutionPolicyCleanUp,
    } = await createLambdaExecutionPolicy({
      name: this.names.LAMBDA_EXECUTION_ROLE_POLICY,
      s3BucketName: this.names.S3_BUCKET,
      iamClient: this.clients.IAM,
      pipelineExecutionRoleArn,
    });
    this.cleanUpFunctions.push(lambdaExecutionPolicyCleanUp);

    console.log(JSON.stringify(lambdaPolicy, null, 2), "\n");

    await this.logger.log(
      MESSAGES.attachPolicy
```

```
        .replace("${POLICY_NAME}", this.names.LAMBDA_EXECUTION_ROLE_POLICY)
        .replace("${ROLE_NAME}", this.names.LAMBDA_EXECUTION_ROLE),
    );

    await this.prompter.checkContinue();

    // Attach the Lambda execution policy to the execution role.
    const { cleanUp: lambdaExecutionRolePolicyCleanUp } = await attachPolicy({
      roleName: this.names.LAMBDA_EXECUTION_ROLE,
      policyArn: lambdaExecutionPolicyArn,
      iamClient: this.clients.IAM,
    });
    this.cleanUpFunctions.push(lambdaExecutionRolePolicyCleanUp);

    await this.logger.log(MESSAGES.policyAttached);

    this.logger.logSeparator();

    // Create Lambda layer for SageMaker packages.
    const { versionArn: layerVersionArn, cleanUp: lambdaLayerCleanUp } =
      await createLambdaLayer({
        name: this.names.LAMBDA_LAYER,
        lambdaClient: this.clients.Lambda,
      });
    this.cleanUpFunctions.push(lambdaLayerCleanUp);

    await this.logger.log(
      MESSAGES.creatingFunction.replace(
        "${FUNCTION_NAME}",
        this.names.LAMBDA_FUNCTION,
      ),
    );

    // Create the Lambda function with the execution role.
    const { arn: lambdaArn, cleanUp: lambdaCleanUp } =
      await createLambdaFunction({
        roleArn: lambdaExecutionRoleArn,
        lambdaClient: this.clients.Lambda,
        name: this.names.LAMBDA_FUNCTION,
        layerVersionArn,
      });
    this.cleanUpFunctions.push(lambdaCleanUp);

    await this.logger.log(
```

```
        MESSAGES.functionCreated.replace(
          "${FUNCTION_NAME}",
          this.names.LAMBDA_FUNCTION,
        ),
      );

      this.logger.logSeparator();

      await this.logger.log(
        MESSAGES.creatingSQSQueue.replace("${QUEUE_NAME}", this.names.SQS_QUEUE),
      );

      // Create an SQS queue for the SageMaker pipeline.
      const {
        queueUrl,
        queueArn,
        cleanUp: queueCleanUp,
      } = await createSQSQueue({
        name: this.names.SQS_QUEUE,
        sqsClient: this.clients.SQS,
      });
      this.cleanUpFunctions.push(queueCleanUp);

      await this.logger.log(
        MESSAGES.sqsQueueCreated.replace("${QUEUE_NAME}", this.names.SQS_QUEUE),
      );

      this.logger.logSeparator();

      await this.logger.log(
        MESSAGES.configuringLambdaSQSEventSource
          .replace("${LAMBDA_NAME}", this.names.LAMBDA_FUNCTION)
          .replace("${QUEUE_NAME}", this.names.SQS_QUEUE),
      );

      // Configure the SQS queue as an event source for the Lambda.
      const { cleanUp: lambdaSQSEventSourceCleanUp } =
        await configureLambdaSQSEventSource({
          lambdaArn,
          lambdaName: this.names.LAMBDA_FUNCTION,
          queueArn,
          sqsClient: this.clients.SQS,
          lambdaClient: this.clients.Lambda,
        });
```

```
    this.cleanUpFunctions.push(lambdaSQSEventSourceCleanUp);

    await this.logger.log(
      MESSAGES.lambdaSQSEventSourceConfigured
        .replace("${LAMBDA_NAME}", this.names.LAMBDA_FUNCTION)
        .replace("${QUEUE_NAME}", this.names.SQS_QUEUE),
    );

    this.logger.logSeparator();

    // Create an IAM policy that allows the SageMaker pipeline to invoke AWS Lambda
    // and send messages to the Amazon SQS queue.
    const {
      arn: pipelineExecutionPolicyArn,
      policy: sagemakerPolicy,
      cleanUp: pipelineExecutionPolicyCleanUp,
    } = await createSagemakerExecutionPolicy({
      sqsQueueArn: queueArn,
      lambdaArn,
      iamClient: this.clients.IAM,
      name: this.names.SAGE_MAKER_EXECUTION_ROLE_POLICY,
      s3BucketName: this.names.S3_BUCKET,
    });
    this.cleanUpFunctions.push(pipelineExecutionPolicyCleanUp);

    console.log(JSON.stringify(sagemakerPolicy, null, 2));

    await this.logger.log(
      MESSAGES.attachPolicy
        .replace("${POLICY_NAME}", this.names.SAGE_MAKER_EXECUTION_ROLE_POLICY)
        .replace("${ROLE_NAME}", this.names.SAGE_MAKER_EXECUTION_ROLE),
    );

    await this.prompter.checkContinue();

    // Attach the SageMaker execution policy to the execution role.
    const { cleanUp: pipelineExecutionRolePolicyCleanUp } = await attachPolicy({
      roleName: this.names.SAGE_MAKER_EXECUTION_ROLE,
      policyArn: pipelineExecutionPolicyArn,
      iamClient: this.clients.IAM,
    });
    this.cleanUpFunctions.push(pipelineExecutionRolePolicyCleanUp);
    // Wait for the role to be ready. If the role is used immediately,
    // the pipeline will fail.
```

```
    await wait(5);

    await this.logger.log(MESSAGES.policyAttached);

    this.logger.logSeparator();

    await this.logger.log(
      MESSAGES.creatingPipeline.replace(
        "${PIPELINE_NAME}",
        this.names.SAGE_MAKER_PIPELINE,
      ),
    );

    // Create the SageMaker pipeline.
    const { cleanUp: pipelineCleanUp } = await createSagemakerPipeline({
      roleArn: pipelineExecutionRoleArn,
      functionArn: lambdaArn,
      sagemakerClient: this.clients.SageMaker,
      name: this.names.SAGE_MAKER_PIPELINE,
    });
    this.cleanUpFunctions.push(pipelineCleanUp);

    await this.logger.log(
      MESSAGES.pipelineCreated.replace(
        "${PIPELINE_NAME}",
        this.names.SAGE_MAKER_PIPELINE,
      ),
    );

    this.logger.logSeparator();

    await this.logger.log(
      MESSAGES.creatingS3Bucket.replace("${BUCKET_NAME}", this.names.S3_BUCKET),
    );

    // Create an S3 bucket for storing inputs and outputs.
    const { cleanUp: s3BucketCleanUp } = await createS3Bucket({
      name: this.names.S3_BUCKET,
      s3Client: this.clients.S3,
    });
    this.cleanUpFunctions.push(s3BucketCleanUp);

    await this.logger.log(
      MESSAGES.s3BucketCreated.replace("${BUCKET_NAME}", this.names.S3_BUCKET),
```

```
    );

    this.logger.logSeparator();

    await this.logger.log(
      MESSAGES.uploadingInputData.replace(
        "${BUCKET_NAME}",
        this.names.S3_BUCKET,
      ),
    );

    // Upload CSV Lat/Long data to S3.
    await uploadCSVDataToS3({
      bucketName: this.names.S3_BUCKET,
      s3Client: this.clients.S3,
    });

    await this.logger.log(MESSAGES.inputDataUploaded);

    this.logger.logSeparator();

    await this.prompter.checkContinue(MESSAGES.executePipeline);

    // Execute the SageMaker pipeline.
    const { arn: pipelineExecutionArn } = await startPipelineExecution({
      name: this.names.SAGE_MAKER_PIPELINE,
      sagemakerClient: this.clients.SageMaker,
      roleArn: pipelineExecutionRoleArn,
      bucketName: this.names.S3_BUCKET,
      queueUrl,
    });

    // Wait for the pipeline execution to finish.
    await waitForPipelineComplete({
      arn: pipelineExecutionArn,
      sagemakerClient: this.clients.SageMaker,
    });

    this.logger.logSeparator();

    await this.logger.log(MESSAGES.outputDelay);

    // The getOutput function will throw an error if the output is not
    // found. The retry function will retry a failed function call once
```

```
    // ever 10 seconds for 2 minutes.
    const output = await retry({ intervalInMs: 10000, maxRetries: 12 }, () =>
      getObject({
        bucket: this.names.S3_BUCKET,
        s3Client: this.clients.S3,
      }),
    );

    this.logger.logSeparator();
    await this.logger.log(MESSAGES.outputDataRetrieved);
    console.log(output.split("\n").slice(0, 6).join("\n"));
  }
}
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考中的以下主题。
  - CreatePipeline
  - DeletePipeline
  - DescribePipelineExecution
  - StartPipelineExecution
  - UpdatePipeline

使用适用于 JavaScript (v3) 的 SDK 的 Secrets Manager 示例

以下代码示例向您展示了如何使用带有 Secrets Manager 的 Amazon SDK for JavaScript (v3) 来执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景 是展示如何通过在同一服务中调用多个函数来完成特定任务的代码示例。

每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关如何在上下文中设置和运行代码的说明。

主题

- 操作

操作

获取密钥值

以下代码示例展示了如何获取 Secrets Manager 密钥值。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import {
  GetSecretValueCommand,
  SecretsManagerClient,
} from "@aws-sdk/client-secrets-manager";

export const getSecretValue = async (secretName = "SECRET_NAME") => {
  const client = new SecretsManagerClient();
  const response = await client.send(
    new GetSecretValueCommand({
      SecretId: secretName,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '584eb612-f8b0-48c9-855e-6d246461b604',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   ARN: 'arn:aws:secretsmanager:us-east-1:xxxxxxxxxxxx:secret:binary-
secret-3873048-xxxxxx',
  //   CreatedDate: 2023-08-08T19:29:51.294Z,
  //   Name: 'binary-secret-3873048',
  //   SecretBinary: Uint8Array(11) [
  //      98, 105, 110, 97, 114,
```

```
  //     121,  32, 100, 97, 116,
  //      97
  //   ],
  //   VersionId: '712083f4-0d26-415e-8044-16735142cd6a',
  //   VersionStages: [ 'AWSCURRENT' ]
  // }

  if (response.SecretString) {
    return response.SecretString;
  }

  if (response.SecretBinary) {
    return response.SecretBinary;
  }
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考GetSecretValue中的。

使用适用于 JavaScript (v3) 的软件开发工具包的 Amazon SES 示例

以下代码示例向您展示如何使用带有 Amazon SES 的 Amazon SDK for JavaScript (v3) 来执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景 是展示如何通过在同一服务中调用多个函数来完成特定任务的代码示例。

每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关如何在上下文中设置和运行代码的说明。

主题

- 操作

操作

创建接收筛选条件

以下代码示例展示了如何创建 Amazon SES 接收筛选条件，以阻止来自某 IP 地址或 IP 地址范围的传入邮件。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import {
  CreateReceiptFilterCommand,
  ReceiptFilterPolicy,
} from "@aws-sdk/client-ses";
import { sesClient } from "./libs/sesClient.js";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";

const createCreateReceiptFilterCommand = ({ policy, ipOrRange, name }) => {
  return new CreateReceiptFilterCommand({
    Filter: {
      IpFilter: {
        Cidr: ipOrRange, // string, either a single IP address (10.0.0.1) or an IP
 address range in CIDR notation (10.0.0.1/24)).
        Policy: policy, // enum ReceiptFilterPolicy, email traffic from the filtered
 addressesOptions.
      },
      /*
        The name of the IP address filter. Only ASCII letters, numbers, underscores,
 or dashes.
        Must be less than 64 characters and start and end with a letter or number.
       */
      Name: name,
    },
  });
};

const FILTER_NAME = getUniqueName("ReceiptFilter");

const run = async () => {
  const createReceiptFilterCommand = createCreateReceiptFilterCommand({
    policy: ReceiptFilterPolicy.Allow,
    ipOrRange: "10.0.0.1",
    name: FILTER_NAME,
  });
```

```
  try {
    return await sesClient.send(createReceiptFilterCommand);
  } catch (err) {
    console.log("Failed to create filter.", err);
    return err;
  }
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考CreateReceiptFilter中的。

创建接收规则

以下代码示例展示了如何创建 Amazon SES 接收规则。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { CreateReceiptRuleCommand, TlsPolicy } from "@aws-sdk/client-ses";
import { sesClient } from "./libs/sesClient.js";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";

const RULE_SET_NAME = getUniqueName("RuleSetName");
const RULE_NAME = getUniqueName("RuleName");
const S3_BUCKET_NAME = getUniqueName("S3BucketName");

const createS3ReceiptRuleCommand = ({
  bucketName,
  emailAddresses,
  name,
  ruleSet,
}) => {
  return new CreateReceiptRuleCommand({
    Rule: {
```

```
      Actions: [
        {
          S3Action: {
            BucketName: bucketName,
            ObjectKeyPrefix: "email",
          },
        },
      ],
      Recipients: emailAddresses,
      Enabled: true,
      Name: name,
      ScanEnabled: false,
      TlsPolicy: TlsPolicy.Optional,
    },
    RuleSetName: ruleSet, // Required
  });
};

const run = async () => {
  const s3ReceiptRuleCommand = createS3ReceiptRuleCommand({
    bucketName: S3_BUCKET_NAME,
    emailAddresses: ["email@example.com"],
    name: RULE_NAME,
    ruleSet: RULE_SET_NAME,
  });

  try {
    return await sesClient.send(s3ReceiptRuleCommand);
  } catch (err) {
    console.log("Failed to create S3 receipt rule.", err);
    throw err;
  }
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考CreateReceiptRule中的。

创建接收规则集

以下代码示例展示了如何创建 Amazon SES 接收规则集，以整理应用到传入邮件的规则。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { CreateReceiptRuleSetCommand } from "@aws-sdk/client-ses";
import { sesClient } from "./libs/sesClient.js";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";

const RULE_SET_NAME = getUniqueName("RuleSetName");

const createCreateReceiptRuleSetCommand = (ruleSetName) => {
  return new CreateReceiptRuleSetCommand({ RuleSetName: ruleSetName });
};

const run = async () => {
  const createReceiptRuleSetCommand =
    createCreateReceiptRuleSetCommand(RULE_SET_NAME);

  try {
    return await sesClient.send(createReceiptRuleSetCommand);
  } catch (err) {
    console.log("Failed to create receipt rule set", err);
    return err;
  }
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考CreateReceiptRuleSet中的。

创建电子邮件模板

以下代码示例展示了如何创建 Amazon SES 电子邮件模板。

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { CreateTemplateCommand } from "@aws-sdk/client-ses";
import { sesClient } from "./libs/sesClient.js";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";

const TEMPLATE_NAME = getUniqueName("TestTemplateName");

const createCreateTemplateCommand = () => {
  return new CreateTemplateCommand({
    /**
     * The template feature in Amazon SES is based on the Handlebars template
 system.
     */
    Template: {
      /**
       * The name of an existing template in Amazon SES.
       */
      TemplateName: TEMPLATE_NAME,
      HtmlPart: `
        <h1>Hello, {{contact.firstName}}!</h1>
        <p>
        Did you know Amazon has a mascot named Peccy?
        </p>
      `,
      SubjectPart: "Amazon Tip",
    },
  });
};

const run = async () => {
  const createTemplateCommand = createCreateTemplateCommand();

  try {
    return await sesClient.send(createTemplateCommand);
```

```
  } catch (err) {
    console.log("Failed to create template.", err);
    return err;
  }
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考CreateTemplate中的。

删除接收筛选条件

以下代码示例展示了如何删除 Amazon SES 接收筛选条件。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { DeleteReceiptFilterCommand } from "@aws-sdk/client-ses";
import { sesClient } from "./libs/sesClient.js";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";

const RECEIPT_FILTER_NAME = getUniqueName("ReceiptFilterName");

const createDeleteReceiptFilterCommand = (filterName) => {
  return new DeleteReceiptFilterCommand({ FilterName: filterName });
};

const run = async () => {
  const deleteReceiptFilterCommand =
    createDeleteReceiptFilterCommand(RECEIPT_FILTER_NAME);

  try {
    return await sesClient.send(deleteReceiptFilterCommand);
  } catch (err) {
    console.log("Error deleting receipt filter.", err);
    return err;
  }
```

```
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DeleteReceiptFilter中的。

删除接收规则

以下代码示例展示了如何删除 Amazon SES 接收规则。

适用于 JavaScript (v3) 的软件开发工具包

> Note
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { DeleteReceiptRuleCommand } from "@aws-sdk/client-ses";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "./libs/sesClient.js";

const RULE_NAME = getUniqueName("RuleName");
const RULE_SET_NAME = getUniqueName("RuleSetName");

const createDeleteReceiptRuleCommand = () => {
  return new DeleteReceiptRuleCommand({
    RuleName: RULE_NAME,
    RuleSetName: RULE_SET_NAME,
  });
};

const run = async () => {
  const deleteReceiptRuleCommand = createDeleteReceiptRuleCommand();
  try {
    return await sesClient.send(deleteReceiptRuleCommand);
  } catch (err) {
    console.log("Failed to delete receipt rule.", err);
    return err;
  }
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DeleteReceiptRule中的。

删除规则集

以下代码示例展示了如何删除 Amazon SES 规则集及其包含的所有规则。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { DeleteReceiptRuleSetCommand } from "@aws-sdk/client-ses";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "./libs/sesClient.js";

const RULE_SET_NAME = getUniqueName("RuleSetName");

const createDeleteReceiptRuleSetCommand = () => {
  return new DeleteReceiptRuleSetCommand({ RuleSetName: RULE_SET_NAME });
};

const run = async () => {
  const deleteReceiptRuleSetCommand = createDeleteReceiptRuleSetCommand();

  try {
    return await sesClient.send(deleteReceiptRuleSetCommand);
  } catch (err) {
    console.log("Failed to delete receipt rule set.", err);
    return err;
  }
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DeleteReceiptRuleSet中的。

删除电子邮件模板

以下代码示例展示了如何删除 Amazon SES 电子邮件模板。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { DeleteTemplateCommand } from "@aws-sdk/client-ses";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "./libs/sesClient.js";

const TEMPLATE_NAME = getUniqueName("TemplateName");

const createDeleteTemplateCommand = (templateName) =>
  new DeleteTemplateCommand({ TemplateName: templateName });

const run = async () => {
  const deleteTemplateCommand = createDeleteTemplateCommand(TEMPLATE_NAME);

  try {
    return await sesClient.send(deleteTemplateCommand);
  } catch (err) {
    console.log("Failed to delete template.", err);
    return err;
  }
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DeleteTemplate中的。

删除身份

以下代码示例展示了如何删除 Amazon SES 身份。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { DeleteIdentityCommand } from "@aws-sdk/client-ses";
import { sesClient } from "./libs/sesClient.js";

const IDENTITY_EMAIL = "fake@example.com";

const createDeleteIdentityCommand = (identityName) => {
  return new DeleteIdentityCommand({
    Identity: identityName,
  });
};

const run = async () => {
  const deleteIdentityCommand = createDeleteIdentityCommand(IDENTITY_EMAIL);

  try {
    return await sesClient.send(deleteIdentityCommand);
  } catch (err) {
    console.log("Failed to delete identity.", err);
    return err;
  }
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DeleteIdentity中的。

获取现有的电子邮件模板

以下代码示例展示了如何获取现有的 Amazon SES 电子邮件模板。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { GetTemplateCommand } from "@aws-sdk/client-ses";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "./libs/sesClient.js";

const TEMPLATE_NAME = getUniqueName("TemplateName");

const createGetTemplateCommand = (templateName) =>
  new GetTemplateCommand({ TemplateName: templateName });

const run = async () => {
  const getTemplateCommand = createGetTemplateCommand(TEMPLATE_NAME);

  try {
    return await sesClient.send(getTemplateCommand);
  } catch (err) {
    console.log("Failed to get email template.", err);
    return err;
  }
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考GetTemplate中的。

列出电子邮件模板

以下代码示例展示了如何列出 Amazon SES 电子邮件模板。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { ListTemplatesCommand } from "@aws-sdk/client-ses";
import { sesClient } from "./libs/sesClient.js";

const createListTemplatesCommand = (maxItems) =>
  new ListTemplatesCommand({ MaxItems: maxItems });

const run = async () => {
  const listTemplatesCommand = createListTemplatesCommand(10);

  try {
    return await sesClient.send(listTemplatesCommand);
  } catch (err) {
    console.log("Failed to list templates.", err);
    return err;
  }
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考ListTemplates中的。

列出身份

以下代码示例展示了如何列出 Amazon SES 身份。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { ListIdentitiesCommand } from "@aws-sdk/client-ses";
import { sesClient } from "./libs/sesClient.js";

const createListIdentitiesCommand = () =>
  new ListIdentitiesCommand({ IdentityType: "EmailAddress", MaxItems: 10 });

const run = async () => {
  const listIdentitiesCommand = createListIdentitiesCommand();

  try {
    return await sesClient.send(listIdentitiesCommand);
  } catch (err) {
    console.log("Failed to list identities.", err);
    return err;
  }
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考ListIdentities中的。

列出接收筛选条件

以下代码示例展示了如何列出 Amazon SES 接收筛选条件。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { ListReceiptFiltersCommand } from "@aws-sdk/client-ses";
import { sesClient } from "./libs/sesClient.js";

const createListReceiptFiltersCommand = () => new ListReceiptFiltersCommand({});

const run = async () => {
  const listReceiptFiltersCommand = createListReceiptFiltersCommand();

  return await sesClient.send(listReceiptFiltersCommand);
```

```
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考ListReceiptFilters中的。

发送批量模板化电子邮件

以下代码示例展示了如何通过 Amazon SES 向多个目的地发送模板化电子邮件。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { SendBulkTemplatedEmailCommand } from "@aws-sdk/client-ses";
import {
  getUniqueName,
  postfix,
} from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "./libs/sesClient.js";

/**
 * Replace this with the name of an existing template.
 */
const TEMPLATE_NAME = getUniqueName("ReminderTemplate");

/**
 * Replace these with existing verified emails.
 */
const VERIFIED_EMAIL_1 = postfix(getUniqueName("Bilbo"), "@example.com");
const VERIFIED_EMAIL_2 = postfix(getUniqueName("Frodo"), "@example.com");

const USERS = [
  { firstName: "Bilbo", emailAddress: VERIFIED_EMAIL_1 },
  { firstName: "Frodo", emailAddress: VERIFIED_EMAIL_2 },
];

/**
 *
```

```
 * @param { { emailAddress: string, firstName: string }[] } users
 * @param { string } templateName the name of an existing template in SES
 * @returns { SendBulkTemplatedEmailCommand }
 */
const createBulkReminderEmailCommand = (users, templateName) => {
  return new SendBulkTemplatedEmailCommand({
    /**
     * Each 'Destination' uses a corresponding set of replacement data. We can map
 each user
     * to a 'Destination' and provide user specific replacement data to create
 personalized emails.
     *
     * Here's an example of how a template would be replaced with user data:
     * Template: <h1>Hello {{name}},</h1><p>Don't forget about the party gifts!</p>
     * Destination 1: <h1>Hello Bilbo,</h1><p>Don't forget about the party gifts!</
p>
     * Destination 2: <h1>Hello Frodo,</h1><p>Don't forget about the party gifts!</
p>
     */
    Destinations: users.map((user) => ({
      Destination: { ToAddresses: [user.emailAddress] },
      ReplacementTemplateData: JSON.stringify({ name: user.firstName }),
    })),
    DefaultTemplateData: JSON.stringify({ name: "Shireling" }),
    Source: VERIFIED_EMAIL_1,
    Template: templateName,
  });
};

const run = async () => {
  const sendBulkTemplateEmailCommand = createBulkReminderEmailCommand(
    USERS,
    TEMPLATE_NAME,
  );
  try {
    return await sesClient.send(sendBulkTemplateEmailCommand);
  } catch (err) {
    console.log("Failed to send bulk template email", err);
    return err;
  }
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考SendBulkTemplatedEmail中的。

发送电子邮件

以下代码示例展示了如何通过 Amazon SES 发送电子邮件。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { SendEmailCommand } from "@aws-sdk/client-ses";
import { sesClient } from "./libs/sesClient.js";

const createSendEmailCommand = (toAddress, fromAddress) => {
  return new SendEmailCommand({
    Destination: {
      /* required */
      CcAddresses: [
        /* more items */
      ],
      ToAddresses: [
        toAddress,
        /* more To-email addresses */
      ],
    },
    Message: {
      /* required */
      Body: {
        /* required */
        Html: {
          Charset: "UTF-8",
          Data: "HTML_FORMAT_BODY",
        },
        Text: {
          Charset: "UTF-8",
          Data: "TEXT_FORMAT_BODY",
        },
```

```
      },
      Subject: {
        Charset: "UTF-8",
        Data: "EMAIL_SUBJECT",
      },
    },
    Source: fromAddress,
    ReplyToAddresses: [
      /* more items */
    ],
  });
};

const run = async () => {
  const sendEmailCommand = createSendEmailCommand(
    "recipient@example.com",
    "sender@example.com",
  );

  try {
    return await sesClient.send(sendEmailCommand);
  } catch (e) {
    console.error("Failed to send email.");
    return e;
  }
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考SendEmail中的。

发送原始电子邮件

以下代码示例展示了如何通过 Amazon SES 发送原始电子邮件。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

使用 nodemailer 发送带附件的电子邮件。

```
import sesClientModule from "@aws-sdk/client-ses";
/**
 * nodemailer wraps the SES SDK and calls SendRawEmail. Use this for more advanced
 * functionality like adding attachments to your email.
 *
 * https://nodemailer.com/transports/ses/
 */
import nodemailer from "nodemailer";

/**
 * @param {string} from An Amazon SES verified email address.
 * @param {*} to An Amazon SES verified email address.
 */
export const sendEmailWithAttachments = (
  from = "from@example.com",
  to = "to@example.com",
) => {
  const ses = new sesClientModule.SESClient({});
  const transporter = nodemailer.createTransport({
    SES: { ses, aws: sesClientModule },
  });

  return new Promise((resolve, reject) => {
    transporter.sendMail(
      {
        from,
        to,
        subject: "Hello World",
        text: "Greetings from Amazon SES!",
        attachments: [{ content: "Hello World!", filename: "hello.txt" }],
      },
      (err, info) => {
        if (err) {
          reject(err);
        } else {
          resolve(info);
        }
      },
    );
  });
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考SendRawEmail中的。

发送模板化电子邮件

以下代码示例展示了如何通过 Amazon SES 发送模板化电子邮件。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { SendTemplatedEmailCommand } from "@aws-sdk/client-ses";
import {
  getUniqueName,
  postfix,
} from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "./libs/sesClient.js";

/**
 * Replace this with the name of an existing template.
 */
const TEMPLATE_NAME = getUniqueName("ReminderTemplate");

/**
 * Replace these with existing verified emails.
 */
const VERIFIED_EMAIL = postfix(getUniqueName("Bilbo"), "@example.com");

const USER = { firstName: "Bilbo", emailAddress: VERIFIED_EMAIL };

/**
 *
 * @param { { emailAddress: string, firstName: string } } user
 * @param { string } templateName - The name of an existing template in Amazon SES.
 * @returns { SendTemplatedEmailCommand }
 */
const createReminderEmailCommand = (user, templateName) => {
```

```
  return new SendTemplatedEmailCommand({
    /**
     * Here's an example of how a template would be replaced with user data:
     * Template: <h1>Hello {{contact.firstName}},</h1><p>Don't forget about the
 party gifts!</p>
     * Destination: <h1>Hello Bilbo,</h1><p>Don't forget about the party gifts!</p>
     */
    Destination: { ToAddresses: [user.emailAddress] },
    TemplateData: JSON.stringify({ contact: { firstName: user.firstName } }),
    Source: VERIFIED_EMAIL,
    Template: templateName,
  });
};

const run = async () => {
  const sendReminderEmailCommand = createReminderEmailCommand(
    USER,
    TEMPLATE_NAME,
  );
  try {
    return await sesClient.send(sendReminderEmailCommand);
  } catch (err) {
    console.log("Failed to send template email", err);
    return err;
  }
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考SendTemplatedEmail中的。

更新电子邮件模板

以下代码示例展示了如何更新 Amazon SES 电子邮件模板。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { UpdateTemplateCommand } from "@aws-sdk/client-ses";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "./libs/sesClient.js";

const TEMPLATE_NAME = getUniqueName("TemplateName");
const HTML_PART = "<h1>Hello, World!</h1>";

const createUpdateTemplateCommand = () => {
  return new UpdateTemplateCommand({
    Template: {
      TemplateName: TEMPLATE_NAME,
      HtmlPart: HTML_PART,
      SubjectPart: "Example",
      TextPart: "Updated template text.",
    },
  });
};

const run = async () => {
  const updateTemplateCommand = createUpdateTemplateCommand();

  try {
    return await sesClient.send(updateTemplateCommand);
  } catch (err) {
    console.log("Failed to update template.", err);
    return err;
  }
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考UpdateTemplate中的。

验证域身份

以下代码示例展示了如何通过 Amazon SES 验证域身份。

适用于 JavaScript (v3) 的软件开发工具包

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { VerifyDomainIdentityCommand } from "@aws-sdk/client-ses";
import {
  getUniqueName,
  postfix,
} from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "./libs/sesClient.js";

/**
 * You must have access to the domain's DNS settings to complete the
 * domain verification process.
 */
const DOMAIN_NAME = postfix(getUniqueName("Domain"), ".example.com");

const createVerifyDomainIdentityCommand = () => {
  return new VerifyDomainIdentityCommand({ Domain: DOMAIN_NAME });
};

const run = async () => {
  const VerifyDomainIdentityCommand = createVerifyDomainIdentityCommand();

  try {
    return await sesClient.send(VerifyDomainIdentityCommand);
  } catch (err) {
    console.log("Failed to verify domain.", err);
    return err;
  }
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考VerifyDomainIdentity中的。

验证电子邮件身份

以下代码示例展示了如何通过 Amazon SES 验证电子邮件身份。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
// Import required AWS SDK clients and commands for Node.js
import { VerifyEmailIdentityCommand } from "@aws-sdk/client-ses";
import { sesClient } from "./libs/sesClient.js";

const EMAIL_ADDRESS = "name@example.com";

const createVerifyEmailIdentityCommand = (emailAddress) => {
  return new VerifyEmailIdentityCommand({ EmailAddress: emailAddress });
};

const run = async () => {
  const verifyEmailIdentityCommand =
    createVerifyEmailIdentityCommand(EMAIL_ADDRESS);
  try {
    return await sesClient.send(verifyEmailIdentityCommand);
  } catch (err) {
    console.log("Failed to verify email identity.", err);
    return err;
  }
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考VerifyEmailIdentity中的。

使用适用于 JavaScript (v3) 的软件开发工具包的亚马逊 SNS 示例

以下代码示例向您展示如何使用带有 Amazon SNS 的 Amazon SDK for JavaScript (v3) 来执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景 是展示如何通过在同一服务中调用多个函数来完成特定任务的代码示例。

每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关如何在上下文中设置和运行代码的说明。

开始使用

开始使用 Amazon SNS

以下代码示例展示了如何开始使用 Amazon SNS。

适用于 JavaScript (v3) 的软件开发工具包

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

初始化 SNS 客户端，并在您的账户中列出主题。

```
import { SNSClient, paginateListTopics } from "@aws-sdk/client-sns";

export const helloSns = async () => {
  // The configuration object (`{}`) is required. If the region and credentials
  // are omitted, the SDK uses your local configuration if it exists.
  const client = new SNSClient({});

  // You can also use `ListTopicsCommand`, but to use that command you must
  // handle the pagination yourself. You can do that by sending the
 `ListTopicsCommand`
  // with the `NextToken` parameter from the previous request.
  const paginatedTopics = paginateListTopics({ client }, {});
  const topics = [];

  for await (const page of paginatedTopics) {
    if (page.Topics?.length) {
      topics.push(...page.Topics);
    }
  }

  const suffix = topics.length === 1 ? "" : "s";

  console.log(
    `Hello, Amazon SNS! You have ${topics.length} topic${suffix} in your account.`,
```

```
  );
  console.log(topics.map((t) => `  * ${t.TopicArn}`).join("\n"));
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考ListTopics中的。

主题

- 操作
- 场景

操作

检查电话号码是否选择不接收消息

以下代码示例展示了如何检查电话号码是否退出接收 Amazon SNS 消息。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

在单独的模块中创建客户端并将其导出。

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

导入 SDK 和客户端模块，然后调用 API。

```
import { CheckIfPhoneNumberIsOptedOutCommand } from "@aws-sdk/client-sns";
```

```
import { snsClient } from "../libs/snsClient.js";

export const checkIfPhoneNumberIsOptedOut = async (
  phoneNumber = "5555555555",
) => {
  const command = new CheckIfPhoneNumberIsOptedOutCommand({
    phoneNumber,
  });

  const response = await snsClient.send(command);
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '3341c28a-cdc8-5b39-a3ee-9fb0ee125732',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   isOptedOut: false
  // }
  return response;
};
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考CheckIfPhoneNumberIsOptedOut中的。

确认端点拥有者想要接收 Amazon SNS 消息

以下代码示例展示了如何通过验证通过较早的订阅操作发送到端点的令牌来确认端点的拥有者想要接收 Amazon SNS 消息。

适用于 JavaScript (v3) 的软件开发工具包

> Note
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

在单独的模块中创建客户端并将其导出。

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

导入 SDK 和客户端模块，然后调用 API。

```
import { ConfirmSubscriptionCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} token - This token is sent the subscriber. Only subscribers
 *                         that are not AWS services (HTTP/S, email) need to be
 confirmed.
 * @param {string} topicArn - The ARN of the topic for which you wish to confirm a
 subscription.
 */
export const confirmSubscription = async (
  token = "TOKEN",
  topicArn = "TOPIC_ARN",
) => {
  const response = await snsClient.send(
    // A subscription only needs to be confirmed if the endpoint type is
    // HTTP/S, email, or in another AWS account.
    new ConfirmSubscriptionCommand({
      Token: token,
      TopicArn: topicArn,
      // If this is true, the subscriber cannot unsubscribe while unauthenticated.
      AuthenticateOnUnsubscribe: "false",
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '4bb5bce9-805a-5517-8333-e1d2cface90b',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
```

```
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   SubscriptionArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:TOPIC_NAME:xxxxxxxx-
xxxx-xxxx-xxxx-xxxxxxxxxxxx'
  // }
  return response;
};
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考ConfirmSubscription中的。

创建主题

以下代码示例展示了如何创建 Amazon SNS 主题。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

在单独的模块中创建客户端并将其导出。

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

导入 SDK 和客户端模块，然后调用 API。

```
import { CreateTopicCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
```

```
 * @param {string} topicName - The name of the topic to create.
 */
export const createTopic = async (topicName = "TOPIC_NAME") => {
  const response = await snsClient.send(
    new CreateTopicCommand({ Name: topicName }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '087b8ad2-4593-50c4-a496-d7e90b82cf3e',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   TopicArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:TOPIC_NAME'
  // }
  return response;
};
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考CreateTopic中的。

删除订阅

以下代码示例展示了如何删除 Amazon SNS 订阅。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

在单独的模块中创建客户端并将其导出。

```
import { SNSClient } from "@aws-sdk/client-sns";
```

```
// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

导入 SDK 和客户端模块，然后调用 API。

```
import { UnsubscribeCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} subscriptionArn - The ARN of the subscription to cancel.
 */
const unsubscribe = async (
  subscriptionArn = "arn:aws:sns:us-east-1:xxxxxxxxxxxx:mytopic:xxxxxxxx-xxxx-xxxx-
xxxx-xxxxxxxxxxxx",
) => {
  const response = await snsClient.send(
    new UnsubscribeCommand({
      SubscriptionArn: subscriptionArn,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '0178259a-9204-507c-b620-78a7570a44c6',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }
  return response;
};
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 详细信息，请参阅《Amazon SDK for JavaScript API 参考》中的 Unsubscribe。

删除主题

以下代码示例展示了如何删除 Amazon SNS 主题以及该主题的所有订阅。

适用于 JavaScript (v3) 的软件开发工具包

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

在单独的模块中创建客户端并将其导出。

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

导入 SDK 和客户端模块，然后调用 API。

```
import { DeleteTopicCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic to delete.
 */
export const deleteTopic = async (topicArn = "TOPIC_ARN") => {
  const response = await snsClient.send(
    new DeleteTopicCommand({ TopicArn: topicArn }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'a10e2886-5a8f-5114-af36-75bd39498332',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
```

```
  //     totalRetryDelay: 0
  //   }
  // }
};
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DeleteTopic中的。

获取主题属性

以下代码示例展示了如何获取 Amazon SNS 主题的属性。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

在单独的模块中创建客户端并将其导出。

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

导入 SDK 和客户端模块，然后调用 API。

```
import { GetTopicAttributesCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic to retrieve attributes for.
 */
export const getTopicAttributes = async (topicArn = "TOPIC_ARN") => {
```

```
  const response = await snsClient.send(
    new GetTopicAttributesCommand({
      TopicArn: topicArn,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '36b6a24e-5473-5d4e-ac32-ff72d9a73d94',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   Attributes: {
  //     Policy: '{...}',
  //     Owner: 'xxxxxxxxxxxx',
  //     SubscriptionsPending: '1',
  //     TopicArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:mytopic',
  //     TracingConfig: 'PassThrough',
  //     EffectiveDeliveryPolicy: '{"http":{"defaultHealthyRetryPolicy":
{"minDelayTarget":20,"maxDelayTarget":20,"numRetries":3,"numMaxDelayRetries":0,"numNoDelayRe
{"headerContentType":"text/plain; charset=UTF-8"}}}',
  //     SubscriptionsConfirmed: '0',
  //     DisplayName: '',
  //     SubscriptionsDeleted: '1'
  //   }
  // }
  return response;
};
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考GetTopicAttributes中的。

适用于 JavaScript (v2) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

导入 SDK 和客户端模块，然后调用 API。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set region
AWS.config.update({ region: "REGION" });

// Create promise and SNS service object
var getTopicAttribsPromise = new AWS.SNS({ apiVersion: "2010-03-31" })
  .getTopicAttributes({ TopicArn: "TOPIC_ARN" })
  .promise();

// Handle promise's fulfilled/rejected states
getTopicAttribsPromise
  .then(function (data) {
    console.log(data);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考GetTopicAttributes中的。

获取发送 SMS 消息的设置

以下代码示例展示了如何获取用于发送 Amazon SNS SMS 消息的设置。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

在单独的模块中创建客户端并将其导出。

```
import { SNSClient } from "@aws-sdk/client-sns";
```

```
// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

导入 SDK 和客户端模块，然后调用 API。

```
import { GetSMSAttributesCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

export const getSmsAttributes = async () => {
  const response = await snsClient.send(
    // If you have not modified the account-level mobile settings of SNS,
    // the DefaultSMSType is undefined. For this example, it was set to
    // Transactional.
    new GetSMSAttributesCommand({ attributes: ["DefaultSMSType"] }),
  );

  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '67ad8386-4169-58f1-bdb9-debd281d48d5',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   attributes: { DefaultSMSType: 'Transactional' }
  // }
  return response;
};
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 详细信息，请参阅《Amazon SDK for JavaScript API 参考》中的 GetSMSAttributes。

列出主题订阅用户

以下代码示例展示了如何检索 Amazon SNS 主题的订阅者列表。

适用于 JavaScript (v3) 的软件开发工具包

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

在单独的模块中创建客户端并将其导出。

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

导入 SDK 和客户端模块，然后调用 API。

```
import { ListSubscriptionsByTopicCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic for which you wish to list
 subscriptions.
 */
export const listSubscriptionsByTopic = async (topicArn = "TOPIC_ARN") => {
  const response = await snsClient.send(
    new ListSubscriptionsByTopicCommand({ TopicArn: topicArn }),
  );

  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '0934fedf-0c4b-572e-9ed2-a3e38fadb0c8',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
```

```
  //   Subscriptions: [
  //     {
  //       SubscriptionArn: 'PendingConfirmation',
  //       Owner: '901487484989',
  //       Protocol: 'email',
  //       Endpoint: 'corepyle@amazon.com',
  //       TopicArn: 'arn:aws:sns:us-east-1:901487484989:mytopic'
  //     }
  //   ]
  // }
  return response;
};
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考ListSubscriptions中的。

列出主题

以下代码示例展示了如何列出 Amazon SNS 主题。

适用于 JavaScript (v3) 的软件开发工具包

> Note
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

在单独的模块中创建客户端并将其导出。

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

导入 SDK 和客户端模块，然后调用 API。

```
import { ListTopicsCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

export const listTopics = async () => {
  const response = await snsClient.send(new ListTopicsCommand({}));
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '936bc5ad-83ca-53c2-b0b7-9891167b909e',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   Topics: [ { TopicArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:mytopic' } ]
  // }
  return response;
};
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考ListTopics中的。

发布带有属性的消息

以下代码示例展示了如何使用 Amazon SNS 发布带有属性的消息。

适用于 JavaScript (v3) 的软件开发工具包

> Note
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

使用组、复制和属性选项向主题发布消息。

```
  async publishMessages() {
    const message = await this.prompter.input({
      message: MESSAGES.publishMessagePrompt,
```

```
    });

    let groupId, deduplicationId, choices;

    if (this.isFifo) {
      await this.logger.log(MESSAGES.groupIdNotice);
      groupId = await this.prompter.input({
        message: MESSAGES.groupIdPrompt,
      });

      if (this.autoDedup === false) {
        await this.logger.log(MESSAGES.deduplicationIdNotice);
        deduplicationId = await this.prompter.input({
          message: MESSAGES.deduplicationIdPrompt,
        });
      }

      choices = await this.prompter.checkbox({
        message: MESSAGES.messageAttributesPrompt,
        choices: toneChoices,
      });
    }

    await this.snsClient.send(
      new PublishCommand({
        TopicArn: this.topicArn,
        Message: message,
        ...(groupId
          ? {
              MessageGroupId: groupId,
            }
          : {}),
        ...(deduplicationId
          ? {
              MessageDeduplicationId: deduplicationId,
            }
          : {}),
        ...(choices
          ? {
              MessageAttributes: {
                tone: {
                  DataType: "String.Array",
                  StringValue: JSON.stringify(choices),
                },
```

```
              },
            }
          : {}),
      }),
    );

    const publishAnother = await this.prompter.confirm({
      message: MESSAGES.publishAnother,
    });

    if (publishAnother) {
      await this.publishMessages();
    }
  }
```

- 有关 API 详细信息，请参阅《Amazon SDK for JavaScript API 参考》中的 Publish。

发布到主题

以下代码示例展示了如何将消息发布到 Amazon SNS 主题。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

在单独的模块中创建客户端并将其导出。

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

导入 SDK 和客户端模块，然后调用 API。

```
import { PublishCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string | Record<string, any>} message - The message to send. Can be a
 plain string or an object
 *                                                 if you are using the `json`
 `MessageStructure`.
 * @param {string} topicArn - The ARN of the topic to which you would like to
 publish.
 */
export const publish = async (
  message = "Hello from SNS!",
  topicArn = "TOPIC_ARN",
) => {
  const response = await snsClient.send(
    new PublishCommand({
      Message: message,
      TopicArn: topicArn,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'e7f77526-e295-5325-9ee4-281a43ad1f05',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   MessageId: 'xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx'
  // }
  return response;
};
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考中的 Publish。

设置发送 SMS 消息的默认设置

以下代码示例展示了如何使用 Amazon SNS 来设置用于发送 SMS 消息的默认设置。

适用于 JavaScript (v3) 的软件开发工具包

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

在单独的模块中创建客户端并将其导出。

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

导入 SDK 和客户端模块，然后调用 API。

```
import { SetSMSAttributesCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {"Transactional" | "Promotional"} defaultSmsType
 */
export const setSmsType = async (defaultSmsType = "Transactional") => {
  const response = await snsClient.send(
    new SetSMSAttributesCommand({
      attributes: {
        // Promotional – (Default) Noncritical messages, such as marketing messages.
        // Transactional – Critical messages that support customer transactions,
        // such as one-time passcodes for multi-factor authentication.
        DefaultSMSType: defaultSmsType,
      },
    }),
  );
  console.log(response);
```

```
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '1885b977-2d7e-535e-8214-e44be727e265',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }
  return response;
};
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考中的 SetSMSAttributes。

设置主题属性

以下代码示例展示了如何设置 Amazon SNS 主题属性。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

在单独的模块中创建客户端并将其导出。

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

导入 SDK 和客户端模块，然后调用 API。

```
import { SetTopicAttributesCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

export const setTopicAttributes = async (
  topicArn = "TOPIC_ARN",
  attributeName = "DisplayName",
  attributeValue = "Test Topic",
) => {
  const response = await snsClient.send(
    new SetTopicAttributesCommand({
      AttributeName: attributeName,
      AttributeValue: attributeValue,
      TopicArn: topicArn,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'd1b08d0e-e9a4-54c3-b8b1-d03238d2b935',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }
  return response;
};
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考SetTopicAttributes中的。

向主题订阅 Lambda 函数

以下代码示例展示了如何订阅 Lambda 函数，以接收来自 Amazon SNS 主题的通知。

适用于 JavaScript (v3) 的软件开发工具包

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

在单独的模块中创建客户端并将其导出。

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

导入 SDK 和客户端模块，然后调用 API。

```
import { SubscribeCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic the subscriber is subscribing to.
 * @param {string} endpoint - The Endpoint ARN of and AWS Lambda function.
 */
export const subscribeLambda = async (
  topicArn = "TOPIC_ARN",
  endpoint = "ENDPOINT",
) => {
  const response = await snsClient.send(
    new SubscribeCommand({
      Protocol: "lambda",
      TopicArn: topicArn,
      Endpoint: endpoint,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
```

```
  //     requestId: 'c8e35bcd-b3c0-5940-9f66-06f6fcc108f0',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   SubscriptionArn: 'pending confirmation'
  // }
  return response;
};
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 详细信息，请参阅《Amazon SDK for JavaScript API 参考》中的 Subscribe。

针对主题订阅移动应用程序

以下代码示例展示了如何订阅移动应用程序端点，以接收来自 Amazon SNS 主题的通知。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

在单独的模块中创建客户端并将其导出。

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

导入 SDK 和客户端模块，然后调用 API。

```
import { SubscribeCommand } from "@aws-sdk/client-sns";
```

```
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic the subscriber is subscribing to.
 * @param {string} endpoint - The Endpoint ARN of an application. This endpoint is
 created
 *                            when an application registers for notifications.
 */
export const subscribeApp = async (
  topicArn = "TOPIC_ARN",
  endpoint = "ENDPOINT",
) => {
  const response = await snsClient.send(
    new SubscribeCommand({
      Protocol: "application",
      TopicArn: topicArn,
      Endpoint: endpoint,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'c8e35bcd-b3c0-5940-9f66-06f6fcc108f0',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   SubscriptionArn: 'pending confirmation'
  // }
  return response;
};
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 详细信息，请参阅 Amazon SDK for JavaScript API 参考中的 Subscribe。

针对主题订阅 SQS 队列

以下代码示例展示了如何订阅 Amazon SQS 队列，以便它接收来自 Amazon SNS 主题的通知。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { SubscribeCommand, SNSClient } from "@aws-sdk/client-sns";

const client = new SNSClient({});

export const subscribeQueue = async (
  topicArn = "TOPIC_ARN",
  queueArn = "QUEUE_ARN",
) => {
  const command = new SubscribeCommand({
    TopicArn: topicArn,
    Protocol: "sqs",
    Endpoint: queueArn,
  });

  const response = await client.send(command);
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '931e13d9-5e2b-543f-8781-4e9e494c5ff2',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   SubscriptionArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:subscribe-queue-
test-430895:xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx'
  // }
  return response;
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考中的 Subscribe。

针对主题订阅电子邮件地址

以下代码示例展示了如何将电子邮件地址订阅到 Amazon SNS 主题。

适用于 JavaScript (v3) 的软件开发工具包

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

在单独的模块中创建客户端并将其导出。

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
 blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

导入 SDK 和客户端模块，然后调用 API。

```
import { SubscribeCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic for which you wish to confirm a
 subscription.
 * @param {string} emailAddress - The email address that is subscribed to the topic.
 */
export const subscribeEmail = async (
  topicArn = "TOPIC_ARN",
  emailAddress = "usern@me.com",
) => {
  const response = await snsClient.send(
    new SubscribeCommand({
      Protocol: "email",
      TopicArn: topicArn,
      Endpoint: emailAddress,
```

```
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'c8e35bcd-b3c0-5940-9f66-06f6fcc108f0',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   SubscriptionArn: 'pending confirmation'
  // }
};
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 详细信息，请参阅 Amazon SDK for JavaScript API 参考中的 Subscribe。

针对主题使用筛选条件进行订阅

以下代码示例展示了如何针对 Amazon SNS 主题使用筛选条件进行订阅。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { SubscribeCommand, SNSClient } from "@aws-sdk/client-sns";

const client = new SNSClient({});

export const subscribeQueueFiltered = async (
  topicArn = "TOPIC_ARN",
  queueArn = "QUEUE_ARN",
) => {
  const command = new SubscribeCommand({
```

```
    TopicArn: topicArn,
    Protocol: "sqs",
    Endpoint: queueArn,
    Attributes: {
      // This subscription will only receive messages with the 'event' attribute set
 to 'order_placed'.
      FilterPolicyScope: "MessageAttributes",
      FilterPolicy: JSON.stringify({
        event: ["order_placed"],
      }),
    },
  });

  const response = await client.send(command);
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '931e13d9-5e2b-543f-8781-4e9e494c5ff2',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   SubscriptionArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:subscribe-queue-
test-430895:xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx'
  // }
  return response;
};
```

- 有关 API 详细信息，请参阅 Amazon SDK for JavaScript API 参考中的 Subscribe。

场景

将消息发布到队列

以下代码示例演示了操作流程：

- 创建主题（FIFO 或非 FIFO）。
- 针对主题订阅多个队列，并提供应用筛选条件的选项。
- 将消息发布到主题。

- 轮询队列中是否有收到的消息。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

这是此工作流程的入口点。

```
import { SNSClient } from "@aws-sdk/client-sns";
import { SQSClient } from "@aws-sdk/client-sqs";

import { TopicsQueuesWkflw } from "./TopicsQueuesWkflw.js";
import { Prompter } from "@aws-doc-sdk-examples/lib/prompter.js";
import { SlowLogger } from "@aws-doc-sdk-examples/lib/slow-logger.js";

export const startSnsWorkflow = () => {
  const noLoggerDelay = process.argv.find((arg) => arg === "--no-logger-delay");
  const snsClient = new SNSClient({});
  const sqsClient = new SQSClient({});
  const prompter = new Prompter();
  const logger = noLoggerDelay ? console : new SlowLogger(25);

  const wkflw = new TopicsQueuesWkflw(snsClient, sqsClient, prompter, logger);

  wkflw.start();
};
```

前面的代码提供必要的依赖关系并启动工作流程。下一节包含示例的大部分内容。

```

const toneChoices = [
  { name: "cheerful", value: "cheerful" },
  { name: "funny", value: "funny" },
  { name: "serious", value: "serious" },
  { name: "sincere", value: "sincere" },
```

```
];

export class TopicsQueuesWkflw {
  // SNS topic is configured as First-In-First-Out
  isFifo = true;

  // Automatic content-based deduplication is enabled.
  autoDedup = false;

  snsClient;
  sqsClient;
  topicName;
  topicArn;
  subscriptionArns = [];
  /**
   * @type {{ queueName: string, queueArn: string, queueUrl: string, policy?:
 string }[]}
   */
  queues = [];
  prompter;

  /**
   * @param {import('@aws-sdk/client-sns').SNSClient} snsClient
   * @param {import('@aws-sdk/client-sqs').SQSClient} sqsClient
   * @param {import('../../libs/prompter.js').Prompter} prompter
   * @param {import('../../libs/logger.js').Logger} logger
   */
  constructor(snsClient, sqsClient, prompter, logger) {
    this.snsClient = snsClient;
    this.sqsClient = sqsClient;
    this.prompter = prompter;
    this.logger = logger;
  }

  async welcome() {
    await this.logger.log(MESSAGES.description);
  }

  async confirmFifo() {
    await this.logger.log(MESSAGES.snsFifoDescription);
    this.isFifo = await this.prompter.confirm({
      message: MESSAGES.snsFifoPrompt,
    });
```

```
    if (this.isFifo) {
      this.logger.logSeparator(MESSAGES.headerDedup);
      await this.logger.log(MESSAGES.deduplicationNotice);
      await this.logger.log(MESSAGES.deduplicationDescription);
      this.autoDedup = await this.prompter.confirm({
        message: MESSAGES.deduplicationPrompt,
      });
    }
  }

  async createTopic() {
    await this.logger.log(MESSAGES.creatingTopics);
    this.topicName = await this.prompter.input({
      message: MESSAGES.topicNamePrompt,
    });
    if (this.isFifo) {
      this.topicName += ".fifo";
      this.logger.logSeparator(MESSAGES.headerFifoNaming);
      await this.logger.log(MESSAGES.appendFifoNotice);
    }

    const response = await this.snsClient.send(
      new CreateTopicCommand({
        Name: this.topicName,
        Attributes: {
          FifoTopic: this.isFifo ? "true" : "false",
          ...(this.autoDedup ? { ContentBasedDeduplication: "true" } : {}),
        },
      }),
    );

    this.topicArn = response.TopicArn;

    await this.logger.log(
      MESSAGES.topicCreatedNotice
        .replace("${TOPIC_NAME}", this.topicName)
        .replace("${TOPIC_ARN}", this.topicArn),
    );
  }

  async createQueues() {
    await this.logger.log(MESSAGES.createQueuesNotice);
    // Increase this number to add more queues.
    let maxQueues = 2;
```

```
    for (let i = 0; i < maxQueues; i++) {
      await this.logger.log(MESSAGES.queueCount.replace("${COUNT}", i + 1));
      let queueName = await this.prompter.input({
        message: MESSAGES.queueNamePrompt.replace(
          "${EXAMPLE_NAME}",
          i === 0 ? "good-news" : "bad-news",
        ),
      });

      if (this.isFifo) {
        queueName += ".fifo";
        await this.logger.log(MESSAGES.appendFifoNotice);
      }

      const response = await this.sqsClient.send(
        new CreateQueueCommand({
          QueueName: queueName,
          Attributes: { ...(this.isFifo ? { FifoQueue: "true" } : {}) },
        }),
      );

      const { Attributes } = await this.sqsClient.send(
        new GetQueueAttributesCommand({
          QueueUrl: response.QueueUrl,
          AttributeNames: ["QueueArn"],
        }),
      );

      this.queues.push({
        queueName,
        queueArn: Attributes.QueueArn,
        queueUrl: response.QueueUrl,
      });

      await this.logger.log(
        MESSAGES.queueCreatedNotice
          .replace("${QUEUE_NAME}", queueName)
          .replace("${QUEUE_URL}", response.QueueUrl)
          .replace("${QUEUE_ARN}", Attributes.QueueArn),
      );
    }
  }
```

```
  async attachQueueIamPolicies() {
    for (const [index, queue] of this.queues.entries()) {
      const policy = JSON.stringify(
        {
          Statement: [
            {
              Effect: "Allow",
              Principal: {
                Service: "sns.amazonaws.com",
              },
              Action: "sqs:SendMessage",
              Resource: queue.queueArn,
              Condition: {
                ArnEquals: {
                  "aws:SourceArn": this.topicArn,
                },
              },
            },
          ],
        },
        null,
        2,
      );

      if (index !== 0) {
        this.logger.logSeparator();
      }

      await this.logger.log(MESSAGES.attachPolicyNotice);
      console.log(policy);
      const addPolicy = await this.prompter.confirm({
        message: MESSAGES.addPolicyConfirmation.replace(
          "${QUEUE_NAME}",
          queue.queueName,
        ),
      });

      if (addPolicy) {
        await this.sqsClient.send(
          new SetQueueAttributesCommand({
            QueueUrl: queue.queueUrl,
            Attributes: {
              Policy: policy,
            },
```

```
          }),
        );
        queue.policy = policy;
      } else {
        await this.logger.log(
          MESSAGES.policyNotAttachedNotice.replace(
            "${QUEUE_NAME}",
            queue.queueName,
          ),
        );
      }
    }
  }

  async subscribeQueuesToTopic() {
    for (const [index, queue] of this.queues.entries()) {
      /**
       * @type {import('@aws-sdk/client-sns').SubscribeCommandInput}
       */
      const subscribeParams = {
        TopicArn: this.topicArn,
        Protocol: "sqs",
        Endpoint: queue.queueArn,
      };
      let tones = [];

      if (this.isFifo) {
        if (index === 0) {
          await this.logger.log(MESSAGES.fifoFilterNotice);
        }
        tones = await this.prompter.checkbox({
          message: MESSAGES.fifoFilterSelect.replace(
            "${QUEUE_NAME}",
            queue.queueName,
          ),
          choices: toneChoices,
        });

        if (tones.length) {
          subscribeParams.Attributes = {
            FilterPolicyScope: "MessageAttributes",
            FilterPolicy: JSON.stringify({
              tone: tones,
            }),
```

```
          };
        }
      }

      const { SubscriptionArn } = await this.snsClient.send(
        new SubscribeCommand(subscribeParams),
      );

      this.subscriptionArns.push(SubscriptionArn);

      await this.logger.log(
        MESSAGES.queueSubscribedNotice
          .replace("${QUEUE_NAME}", queue.queueName)
          .replace("${TOPIC_NAME}", this.topicName)
          .replace("${TONES}", tones.length ? tones.join(", ") : "none"),
      );
    }
  }

  async publishMessages() {
    const message = await this.prompter.input({
      message: MESSAGES.publishMessagePrompt,
    });

    let groupId, deduplicationId, choices;

    if (this.isFifo) {
      await this.logger.log(MESSAGES.groupIdNotice);
      groupId = await this.prompter.input({
        message: MESSAGES.groupIdPrompt,
      });

      if (this.autoDedup === false) {
        await this.logger.log(MESSAGES.deduplicationIdNotice);
        deduplicationId = await this.prompter.input({
          message: MESSAGES.deduplicationIdPrompt,
        });
      }

      choices = await this.prompter.checkbox({
        message: MESSAGES.messageAttributesPrompt,
        choices: toneChoices,
      });
    }
```

```
    await this.snsClient.send(
      new PublishCommand({
        TopicArn: this.topicArn,
        Message: message,
        ...(groupId
          ? {
              MessageGroupId: groupId,
            }
          : {}),
        ...(deduplicationId
          ? {
              MessageDeduplicationId: deduplicationId,
            }
          : {}),
        ...(choices
          ? {
              MessageAttributes: {
                tone: {
                  DataType: "String.Array",
                  StringValue: JSON.stringify(choices),
                },
              },
            }
          : {}),
      }),
    );

    const publishAnother = await this.prompter.confirm({
      message: MESSAGES.publishAnother,
    });

    if (publishAnother) {
      await this.publishMessages();
    }
  }

  async receiveAndDeleteMessages() {
    for (const queue of this.queues) {
      const { Messages } = await this.sqsClient.send(
        new ReceiveMessageCommand({
          QueueUrl: queue.queueUrl,
        }),
      );
```

```
      if (Messages) {
        await this.logger.log(
          MESSAGES.messagesReceivedNotice.replace(
            "${QUEUE_NAME}",
            queue.queueName,
          ),
        );
        console.log(Messages);

        await this.sqsClient.send(
          new DeleteMessageBatchCommand({
            QueueUrl: queue.queueUrl,
            Entries: Messages.map((message) => ({
              Id: message.MessageId,
              ReceiptHandle: message.ReceiptHandle,
            })),
          }),
        );
      } else {
        await this.logger.log(
          MESSAGES.noMessagesReceivedNotice.replace(
            "${QUEUE_NAME}",
            queue.queueName,
          ),
        );
      }
    }

    const deleteAndPoll = await this.prompter.confirm({
      message: MESSAGES.deleteAndPollConfirmation,
    });

    if (deleteAndPoll) {
      await this.receiveAndDeleteMessages();
    }
  }

  async destroyResources() {
    for (const subscriptionArn of this.subscriptionArns) {
      await this.snsClient.send(
        new UnsubscribeCommand({ SubscriptionArn: subscriptionArn }),
      );
    }
```

```
    for (const queue of this.queues) {
      await this.sqsClient.send(
        new DeleteQueueCommand({ QueueUrl: queue.queueUrl }),
      );
    }

    if (this.topicArn) {
      await this.snsClient.send(
        new DeleteTopicCommand({ TopicArn: this.topicArn }),
      );
    }
  }

  async start() {
    console.clear();

    try {
      this.logger.logSeparator(MESSAGES.headerWelcome);
      await this.welcome();
      this.logger.logSeparator(MESSAGES.headerFifo);
      await this.confirmFifo();
      this.logger.logSeparator(MESSAGES.headerCreateTopic);
      await this.createTopic();
      this.logger.logSeparator(MESSAGES.headerCreateQueues);
      await this.createQueues();
      this.logger.logSeparator(MESSAGES.headerAttachPolicy);
      await this.attachQueueIamPolicies();
      this.logger.logSeparator(MESSAGES.headerSubscribeQueues);
      await this.subscribeQueuesToTopic();
      this.logger.logSeparator(MESSAGES.headerPublishMessage);
      await this.publishMessages();
      this.logger.logSeparator(MESSAGES.headerReceiveMessages);
      await this.receiveAndDeleteMessages();
    } catch (err) {
      console.error(err);
    } finally {
      await this.destroyResources();
    }
  }
}
```

- 有关 API 详细信息，请参阅《Amazon SDK for JavaScript API 参考》中的以下主题。

- CreateQueue
- CreateTopic
- DeleteMessageBatch
- DeleteQueue
- DeleteTopic
- GetQueueAttributes
- Publish
- ReceiveMessage
- SetQueueAttributes
- 订阅
- Unsubscribe

使用适用于 JavaScript (v3) 的软件开发工具包的亚马逊 SQS 示例

以下代码示例向您展示如何使用带有 Amazon SQS 的 Amazon SDK for JavaScript (v3) 来执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景 是展示如何通过在同一服务中调用多个函数来完成特定任务的代码示例。

每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关如何在上下文中设置和运行代码的说明。

开始使用

Hello Amazon SQS

以下代码示例展示了如何开始使用 Amazon SQS。

适用于 JavaScript (v3) 的软件开发工具包

Note

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

初始化 Amazon SQS 客户端并列出队列。

```
import { SQSClient, paginateListQueues } from "@aws-sdk/client-sqs";

export const helloSqs = async () => {
  // The configuration object (`{}`) is required. If the region and credentials
  // are omitted, the SDK uses your local configuration if it exists.
  const client = new SQSClient({});

  // You can also use `ListQueuesCommand`, but to use that command you must
  // handle the pagination yourself. You can do that by sending the
 `ListQueuesCommand`
  // with the `NextToken` parameter from the previous request.
  const paginatedQueues = paginateListQueues({ client }, {});
  const queues = [];

  for await (const page of paginatedQueues) {
    if (page.QueueUrls?.length) {
      queues.push(...page.QueueUrls);
    }
  }

  const suffix = queues.length === 1 ? "" : "s";

  console.log(
    `Hello, Amazon SQS! You have ${queues.length} queue${suffix} in your account.`,
  );
  console.log(queues.map((t) => `  * ${t}`).join("\n"));
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考ListQueues中的。

主题

- 操作
- 场景

操作

更改消息超时可见性

以下代码示例展示了如何更改 Amazon SQS 消息超时可见性。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

接收 Amazon SQS 消息并更改其超时可见性。

```
import {
  ReceiveMessageCommand,
  ChangeMessageVisibilityCommand,
  SQSClient,
} from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue_url";

const receiveMessage = (queueUrl) =>
  client.send(
    new ReceiveMessageCommand({
      AttributeNames: ["SentTimestamp"],
      MaxNumberOfMessages: 1,
      MessageAttributeNames: ["All"],
      QueueUrl: queueUrl,
      WaitTimeSeconds: 1,
    }),
  );

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const { Messages } = await receiveMessage(queueUrl);

  const response = await client.send(
    new ChangeMessageVisibilityCommand({
      QueueUrl: queueUrl,
```

```
      ReceiptHandle: Messages[0].ReceiptHandle,
      VisibilityTimeout: 20,
    }),
  );
  console.log(response);
  return response;
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考ChangeMessageVisibility中的。

适用于 JavaScript (v2) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

接收 Amazon SQS 消息并更改其超时可见性。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region to us-west-2
AWS.config.update({ region: "us-west-2" });

// Create the SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var queueURL = "https://sqs.REGION.amazonaws.com/ACCOUNT-ID/QUEUE-NAME";

var params = {
  AttributeNames: ["SentTimestamp"],
  MaxNumberOfMessages: 1,
  MessageAttributeNames: ["All"],
  QueueUrl: queueURL,
};

sqs.receiveMessage(params, function (err, data) {
  if (err) {
    console.log("Receive Error", err);
  } else {
```

```
    // Make sure we have a message
    if (data.Messages != null) {
      var visibilityParams = {
        QueueUrl: queueURL,
        ReceiptHandle: data.Messages[0].ReceiptHandle,
        VisibilityTimeout: 20, // 20 second timeout
      };
      sqs.changeMessageVisibility(visibilityParams, function (err, data) {
        if (err) {
          console.log("Delete Error", err);
        } else {
          console.log("Timeout Changed", data);
        }
      });
    } else {
      console.log("No messages to change");
    }
  }
});
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考ChangeMessageVisibility中的。

配置死信队列

以下代码示例展示了如何在 Amazon SQS 中配置死信队列。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { SetQueueAttributesCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue_url";
```

```
const DEAD_LETTER_QUEUE_ARN = "dead_letter_queue_arn";

export const main = async (
  queueUrl = SQS_QUEUE_URL,
  deadLetterQueueArn = DEAD_LETTER_QUEUE_ARN,
) => {
  const command = new SetQueueAttributesCommand({
    Attributes: {
      RedrivePolicy: JSON.stringify({
        // Amazon SQS supports dead-letter queues (DLQ), which other
        // queues (source queues) can target for messages that can't
        // be processed (consumed) successfully.
        // https://docs.aws.amazon.com/AWSSimpleQueueService/latest/
SQSDeveloperGuide/sqs-dead-letter-queues.html
        deadLetterTargetArn: deadLetterQueueArn,
        maxReceiveCount: "10",
      }),
    },
    QueueUrl: queueUrl,
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考SetQueueAttributes中的。

创建队列

以下代码示例展示了如何创建 Amazon SQS 队列。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

创建 Amazon SQS 标准队列。

```
import { CreateQueueCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_NAME = "test-queue";

export const main = async (sqsQueueName = SQS_QUEUE_NAME) => {
  const command = new CreateQueueCommand({
    QueueName: sqsQueueName,
    Attributes: {
      DelaySeconds: "60",
      MessageRetentionPeriod: "86400",
    },
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

使用长轮询创建 Amazon SQS 队列。

```
import { CreateQueueCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_NAME = "queue_name";

export const main = async (queueName = SQS_QUEUE_NAME) => {
  const response = await client.send(
    new CreateQueueCommand({
      QueueName: queueName,
      Attributes: {
        // When the wait time for the ReceiveMessage API action is greater than 0,
        // long polling is in effect. The maximum long polling wait time is 20
        // seconds. Long polling helps reduce the cost of using Amazon SQS by,
        // eliminating the number of empty responses and false empty responses.
        // https://docs.aws.amazon.com/AWSSimpleQueueService/latest/
SQSDeveloperGuide/sqs-short-and-long-polling.html
        ReceiveMessageWaitTimeSeconds: "20",
      },
    }),
  );
  console.log(response);
```

```
  return response;
};
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考CreateQueue中的。

适用于 JavaScript (v2) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

创建 Amazon SQS 标准队列。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create an SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var params = {
  QueueName: "SQS_QUEUE_NAME",
  Attributes: {
    DelaySeconds: "60",
    MessageRetentionPeriod: "86400",
  },
};

sqs.createQueue(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.QueueUrl);
  }
});
```

创建等待消息到达的 Amazon SQS 队列。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var params = {
  QueueName: "SQS_QUEUE_NAME",
  Attributes: {
    ReceiveMessageWaitTimeSeconds: "20",
  },
};

sqs.createQueue(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.QueueUrl);
  }
});
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考CreateQueue中的。

从队列中删除一批消息

以下代码示例展示了如何从 Amazon SQS 队列中删除一批消息。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import {
  ReceiveMessageCommand,
  DeleteMessageCommand,
  SQSClient,
  DeleteMessageBatchCommand,
} from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue_url";

const receiveMessage = (queueUrl) =>
  client.send(
    new ReceiveMessageCommand({
      AttributeNames: ["SentTimestamp"],
      MaxNumberOfMessages: 10,
      MessageAttributeNames: ["All"],
      QueueUrl: queueUrl,
      WaitTimeSeconds: 20,
      VisibilityTimeout: 20,
    }),
  );

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const { Messages } = await receiveMessage(queueUrl);

  if (!Messages) {
    return;
  }

  if (Messages.length === 1) {
    console.log(Messages[0].Body);
    await client.send(
      new DeleteMessageCommand({
        QueueUrl: queueUrl,
        ReceiptHandle: Messages[0].ReceiptHandle,
      }),
    );
  } else {
    await client.send(
      new DeleteMessageBatchCommand({
        QueueUrl: queueUrl,
        Entries: Messages.map((message) => ({
          Id: message.MessageId,
```

```
          ReceiptHandle: message.ReceiptHandle,
        })),
      }),
    );
  }
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DeleteMessageBatch中的。

从队列中删除消息

以下代码示例展示了如何从 Amazon SQS 队列中删除消息。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

接收和删除 Amazon SQS 消息。

```
import {
  ReceiveMessageCommand,
  DeleteMessageCommand,
  SQSClient,
  DeleteMessageBatchCommand,
} from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue_url";

const receiveMessage = (queueUrl) =>
  client.send(
    new ReceiveMessageCommand({
      AttributeNames: ["SentTimestamp"],
      MaxNumberOfMessages: 10,
      MessageAttributeNames: ["All"],
      QueueUrl: queueUrl,
```

```
      WaitTimeSeconds: 20,
      VisibilityTimeout: 20,
    }),
  );

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const { Messages } = await receiveMessage(queueUrl);

  if (!Messages) {
    return;
  }

  if (Messages.length === 1) {
    console.log(Messages[0].Body);
    await client.send(
      new DeleteMessageCommand({
        QueueUrl: queueUrl,
        ReceiptHandle: Messages[0].ReceiptHandle,
      }),
    );
  } else {
    await client.send(
      new DeleteMessageBatchCommand({
        QueueUrl: queueUrl,
        Entries: Messages.map((message) => ({
          Id: message.MessageId,
          ReceiptHandle: message.ReceiptHandle,
        })),
      }),
    );
  }
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DeleteMessage中的。

适用于 JavaScript (v2) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

接收和删除 Amazon SQS 消息。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create an SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var queueURL = "SQS_QUEUE_URL";

var params = {
  AttributeNames: ["SentTimestamp"],
  MaxNumberOfMessages: 10,
  MessageAttributeNames: ["All"],
  QueueUrl: queueURL,
  VisibilityTimeout: 20,
  WaitTimeSeconds: 0,
};

sqs.receiveMessage(params, function (err, data) {
  if (err) {
    console.log("Receive Error", err);
  } else if (data.Messages) {
    var deleteParams = {
      QueueUrl: queueURL,
      ReceiptHandle: data.Messages[0].ReceiptHandle,
    };
    sqs.deleteMessage(deleteParams, function (err, data) {
      if (err) {
        console.log("Delete Error", err);
      } else {
        console.log("Message Deleted", data);
      }
    });
  }
});
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DeleteMessage中的。

删除队列

以下代码示例展示了如何删除 Amazon SQS 队列。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

删除 Amazon SQS 队列。

```
import { DeleteQueueCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "test-queue-url";

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const command = new DeleteQueueCommand({ QueueUrl: queueUrl });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DeleteQueue中的。

适用于 JavaScript (v2) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

删除 Amazon SQS 队列。

```
// Load the AWS SDK for Node.js
```

```
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create an SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var params = {
  QueueUrl: "SQS_QUEUE_URL",
};

sqs.deleteQueue(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DeleteQueue中的。

获取队列的属性

以下代码示例展示了如何从 Amazon SQS 队列中获取属性。

适用于 JavaScript (v3) 的软件开发工具包

> Note
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { GetQueueAttributesCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue-url";

export const getQueueAttributes = async (queueUrl = SQS_QUEUE_URL) => {
```

```
  const command = new GetQueueAttributesCommand({
    QueueUrl: queueUrl,
    AttributeNames: ["DelaySeconds"],
  });

  const response = await client.send(command);
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '747a1192-c334-5682-a508-4cd5e8dc4e79',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   Attributes: { DelaySeconds: '1' }
  // }
  return response;
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考GetQueueAttributes中的。

获取队列的 URL

以下代码示例展示了如何获取 Amazon SQS 队列的 URL。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

获取 Amazon SQS 队列的 URL。

```
import { GetQueueUrlCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_NAME = "test-queue";
```

```
export const main = async (queueName = SQS_QUEUE_NAME) => {
  const command = new GetQueueUrlCommand({ QueueName: queueName });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考GetQueueUrl中的。

适用于 JavaScript (v2) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

获取 Amazon SQS 队列的 URL。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create an SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var params = {
  QueueName: "SQS_QUEUE_NAME",
};

sqs.getQueueUrl(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.QueueUrl);
  }
});
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考GetQueueUrl中的。

列出队列

以下代码示例展示了如何列出 Amazon SQS 队列。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

列出 Amazon SQS 队列。

```
import { paginateListQueues, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});

export const main = async () => {
  const paginatedListQueues = paginateListQueues({ client }, {});

  /** @type {string[]} */
  const urls = [];
  for await (const page of paginatedListQueues) {
    const nextUrls = page.QueueUrls?.filter((qurl) => !!qurl) || [];
    urls.push(...nextUrls);
    urls.forEach((url) => console.log(url));
  }

  return urls;
};
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考ListQueues中的。

适用于 JavaScript (v2) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

列出 Amazon SQS 队列。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create an SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var params = {};

sqs.listQueues(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.QueueUrls);
  }
});
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考ListQueues中的。

接收来自队列的消息

以下代码示例展示了如何从 Amazon SQS 队列中接收消息。

适用于 JavaScript (v3) 的软件开发工具包

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

接收来自 Amazon SQS 队列的消息。

```
import {
  ReceiveMessageCommand,
  DeleteMessageCommand,
  SQSClient,
  DeleteMessageBatchCommand,
} from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue_url";

const receiveMessage = (queueUrl) =>
  client.send(
    new ReceiveMessageCommand({
      AttributeNames: ["SentTimestamp"],
      MaxNumberOfMessages: 10,
      MessageAttributeNames: ["All"],
      QueueUrl: queueUrl,
      WaitTimeSeconds: 20,
      VisibilityTimeout: 20,
    }),
  );

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const { Messages } = await receiveMessage(queueUrl);

  if (!Messages) {
    return;
  }

  if (Messages.length === 1) {
    console.log(Messages[0].Body);
    await client.send(
      new DeleteMessageCommand({
```

```
        QueueUrl: queueUrl,
        ReceiptHandle: Messages[0].ReceiptHandle,
      }),
    );
  } else {
    await client.send(
      new DeleteMessageBatchCommand({
        QueueUrl: queueUrl,
        Entries: Messages.map((message) => ({
          Id: message.MessageId,
          ReceiptHandle: message.ReceiptHandle,
        })),
      }),
    );
  }
};
```

使用长轮询支持接收来自 Amazon SQS 队列的消息。

```
import { ReceiveMessageCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue-url";

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const command = new ReceiveMessageCommand({
    AttributeNames: ["SentTimestamp"],
    MaxNumberOfMessages: 1,
    MessageAttributeNames: ["All"],
    QueueUrl: queueUrl,
    // The duration (in seconds) for which the call waits for a message
    // to arrive in the queue before returning. If a message is available,
    // the call returns sooner than WaitTimeSeconds. If no messages are
    // available and the wait time expires, the call returns successfully
    // with an empty list of messages.
    // https://docs.aws.amazon.com/AWSSimpleQueueService/latest/APIReference/
API_ReceiveMessage.html#API_ReceiveMessage_RequestSyntax
    WaitTimeSeconds: 20,
  });

  const response = await client.send(command);
  console.log(response);
```

```
  return response;
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考ReceiveMessage中的。

适用于 JavaScript (v2) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

使用长轮询接收来自 Amazon SQS 队列的消息。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var queueURL = "SQS_QUEUE_URL";

var params = {
  AttributeNames: ["SentTimestamp"],
  MaxNumberOfMessages: 1,
  MessageAttributeNames: ["All"],
  QueueUrl: queueURL,
  WaitTimeSeconds: 20,
};

sqs.receiveMessage(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考ReceiveMessage中的。

向队列发送消息

以下代码示例展示了如何向 Amazon SQS 队列发送消息。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

向 Amazon SQS 队列发送消息。

```
import { SendMessageCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue_url";

export const main = async (sqsQueueUrl = SQS_QUEUE_URL) => {
  const command = new SendMessageCommand({
    QueueUrl: sqsQueueUrl,
    DelaySeconds: 10,
    MessageAttributes: {
      Title: {
        DataType: "String",
        StringValue: "The Whistler",
      },
      Author: {
        DataType: "String",
        StringValue: "John Grisham",
      },
      WeeksOn: {
        DataType: "Number",
        StringValue: "6",
      },
    },
    MessageBody:
```

```
      "Information about current NY Times fiction bestseller for week of
 12/11/2016.",
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考SendMessage中的。

适用于 JavaScript (v2) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

向 Amazon SQS 队列发送消息。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create an SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var params = {
  // Remove DelaySeconds parameter and value for FIFO queues
  DelaySeconds: 10,
  MessageAttributes: {
    Title: {
      DataType: "String",
      StringValue: "The Whistler",
    },
    Author: {
      DataType: "String",
      StringValue: "John Grisham",
    },
```

```
    WeeksOn: {
      DataType: "Number",
      StringValue: "6",
    },
  },
  MessageBody:
    "Information about current NY Times fiction bestseller for week of 12/11/2016.",
  // MessageDeduplicationId: "TheWhistler",  // Required for FIFO queues
  // MessageGroupId: "Group1",  // Required for FIFO queues
  QueueUrl: "SQS_QUEUE_URL",
};

sqs.sendMessage(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.MessageId);
  }
});
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考SendMessage中的。

设置队列属性

以下代码示例展示了如何为 Amazon SQS 队列设置属性。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { SetQueueAttributesCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue-url";
```

```
export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const command = new SetQueueAttributesCommand({
    QueueUrl: queueUrl,
    Attributes: {
      DelaySeconds: "1",
    },
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

将 Amazon SQS 队列配置为使用长轮询。

```
import { SetQueueAttributesCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue_url";

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const command = new SetQueueAttributesCommand({
    Attributes: {
      ReceiveMessageWaitTimeSeconds: "20",
    },
    QueueUrl: queueUrl,
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考SetQueueAttributes中的。

场景

将消息发布到队列

以下代码示例演示了操作流程：

- 创建主题（FIFO 或非 FIFO）。
- 针对主题订阅多个队列，并提供应用筛选条件的选项。
- 将消息发布到主题。
- 轮询队列中是否有收到的消息。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

这是此工作流程的入口点。

```
import { SNSClient } from "@aws-sdk/client-sns";
import { SQSClient } from "@aws-sdk/client-sqs";

import { TopicsQueuesWkflw } from "./TopicsQueuesWkflw.js";
import { Prompter } from "@aws-doc-sdk-examples/lib/prompter.js";
import { SlowLogger } from "@aws-doc-sdk-examples/lib/slow-logger.js";

export const startSnsWorkflow = () => {
  const noLoggerDelay = process.argv.find((arg) => arg === "--no-logger-delay");
  const snsClient = new SNSClient({});
  const sqsClient = new SQSClient({});
  const prompter = new Prompter();
  const logger = noLoggerDelay ? console : new SlowLogger(25);

  const wkflw = new TopicsQueuesWkflw(snsClient, sqsClient, prompter, logger);

  wkflw.start();
};
```

前面的代码提供必要的依赖关系并启动工作流程。下一节包含示例的大部分内容。

```
const toneChoices = [
```

```
  { name: "cheerful", value: "cheerful" },
  { name: "funny", value: "funny" },
  { name: "serious", value: "serious" },
  { name: "sincere", value: "sincere" },
];

export class TopicsQueuesWkflw {
  // SNS topic is configured as First-In-First-Out
  isFifo = true;

  // Automatic content-based deduplication is enabled.
  autoDedup = false;

  snsClient;
  sqsClient;
  topicName;
  topicArn;
  subscriptionArns = [];
  /**
   * @type {{ queueName: string, queueArn: string, queueUrl: string, policy?:
 string }[]}
   */
  queues = [];
  prompter;

  /**
   * @param {import('@aws-sdk/client-sns').SNSClient} snsClient
   * @param {import('@aws-sdk/client-sqs').SQSClient} sqsClient
   * @param {import('../../libs/prompter.js').Prompter} prompter
   * @param {import('../../libs/logger.js').Logger} logger
   */
  constructor(snsClient, sqsClient, prompter, logger) {
    this.snsClient = snsClient;
    this.sqsClient = sqsClient;
    this.prompter = prompter;
    this.logger = logger;
  }

  async welcome() {
    await this.logger.log(MESSAGES.description);
  }

  async confirmFifo() {
    await this.logger.log(MESSAGES.snsFifoDescription);
```

```
    this.isFifo = await this.prompter.confirm({
      message: MESSAGES.snsFifoPrompt,
    });

    if (this.isFifo) {
      this.logger.logSeparator(MESSAGES.headerDedup);
      await this.logger.log(MESSAGES.deduplicationNotice);
      await this.logger.log(MESSAGES.deduplicationDescription);
      this.autoDedup = await this.prompter.confirm({
        message: MESSAGES.deduplicationPrompt,
      });
    }
  }

  async createTopic() {
    await this.logger.log(MESSAGES.creatingTopics);
    this.topicName = await this.prompter.input({
      message: MESSAGES.topicNamePrompt,
    });
    if (this.isFifo) {
      this.topicName += ".fifo";
      this.logger.logSeparator(MESSAGES.headerFifoNaming);
      await this.logger.log(MESSAGES.appendFifoNotice);
    }

    const response = await this.snsClient.send(
      new CreateTopicCommand({
        Name: this.topicName,
        Attributes: {
          FifoTopic: this.isFifo ? "true" : "false",
          ...(this.autoDedup ? { ContentBasedDeduplication: "true" } : {}),
        },
      }),
    );

    this.topicArn = response.TopicArn;

    await this.logger.log(
      MESSAGES.topicCreatedNotice
        .replace("${TOPIC_NAME}", this.topicName)
        .replace("${TOPIC_ARN}", this.topicArn),
    );
  }
```

```
  async createQueues() {
    await this.logger.log(MESSAGES.createQueuesNotice);
    // Increase this number to add more queues.
    let maxQueues = 2;

    for (let i = 0; i < maxQueues; i++) {
      await this.logger.log(MESSAGES.queueCount.replace("${COUNT}", i + 1));
      let queueName = await this.prompter.input({
        message: MESSAGES.queueNamePrompt.replace(
          "${EXAMPLE_NAME}",
          i === 0 ? "good-news" : "bad-news",
        ),
      });

      if (this.isFifo) {
        queueName += ".fifo";
        await this.logger.log(MESSAGES.appendFifoNotice);
      }

      const response = await this.sqsClient.send(
        new CreateQueueCommand({
          QueueName: queueName,
          Attributes: { ...(this.isFifo ? { FifoQueue: "true" } : {}) },
        }),
      );

      const { Attributes } = await this.sqsClient.send(
        new GetQueueAttributesCommand({
          QueueUrl: response.QueueUrl,
          AttributeNames: ["QueueArn"],
        }),
      );

      this.queues.push({
        queueName,
        queueArn: Attributes.QueueArn,
        queueUrl: response.QueueUrl,
      });

      await this.logger.log(
        MESSAGES.queueCreatedNotice
          .replace("${QUEUE_NAME}", queueName)
          .replace("${QUEUE_URL}", response.QueueUrl)
          .replace("${QUEUE_ARN}", Attributes.QueueArn),
```

```
      );
    }
  }

  async attachQueueIamPolicies() {
    for (const [index, queue] of this.queues.entries()) {
      const policy = JSON.stringify(
        {
          Statement: [
            {
              Effect: "Allow",
              Principal: {
                Service: "sns.amazonaws.com",
              },
              Action: "sqs:SendMessage",
              Resource: queue.queueArn,
              Condition: {
                ArnEquals: {
                  "aws:SourceArn": this.topicArn,
                },
              },
            },
          ],
        },
        null,
        2,
      );

      if (index !== 0) {
        this.logger.logSeparator();
      }

      await this.logger.log(MESSAGES.attachPolicyNotice);
      console.log(policy);
      const addPolicy = await this.prompter.confirm({
        message: MESSAGES.addPolicyConfirmation.replace(
          "${QUEUE_NAME}",
          queue.queueName,
        ),
      });

      if (addPolicy) {
        await this.sqsClient.send(
          new SetQueueAttributesCommand({
```

```
          QueueUrl: queue.queueUrl,
          Attributes: {
            Policy: policy,
          },
        }),
      );
      queue.policy = policy;
    } else {
      await this.logger.log(
        MESSAGES.policyNotAttachedNotice.replace(
          "${QUEUE_NAME}",
          queue.queueName,
        ),
      );
    }
  }
}

async subscribeQueuesToTopic() {
  for (const [index, queue] of this.queues.entries()) {
    /**
     * @type {import('@aws-sdk/client-sns').SubscribeCommandInput}
     */
    const subscribeParams = {
      TopicArn: this.topicArn,
      Protocol: "sqs",
      Endpoint: queue.queueArn,
    };
    let tones = [];

    if (this.isFifo) {
      if (index === 0) {
        await this.logger.log(MESSAGES.fifoFilterNotice);
      }
      tones = await this.prompter.checkbox({
        message: MESSAGES.fifoFilterSelect.replace(
          "${QUEUE_NAME}",
          queue.queueName,
        ),
        choices: toneChoices,
      });

      if (tones.length) {
        subscribeParams.Attributes = {
```

```
            FilterPolicyScope: "MessageAttributes",
            FilterPolicy: JSON.stringify({
              tone: tones,
            }),
          };
        }
      }

      const { SubscriptionArn } = await this.snsClient.send(
        new SubscribeCommand(subscribeParams),
      );

      this.subscriptionArns.push(SubscriptionArn);

      await this.logger.log(
        MESSAGES.queueSubscribedNotice
          .replace("${QUEUE_NAME}", queue.queueName)
          .replace("${TOPIC_NAME}", this.topicName)
          .replace("${TONES}", tones.length ? tones.join(", ") : "none"),
      );
    }
  }

  async publishMessages() {
    const message = await this.prompter.input({
      message: MESSAGES.publishMessagePrompt,
    });

    let groupId, deduplicationId, choices;

    if (this.isFifo) {
      await this.logger.log(MESSAGES.groupIdNotice);
      groupId = await this.prompter.input({
        message: MESSAGES.groupIdPrompt,
      });

      if (this.autoDedup === false) {
        await this.logger.log(MESSAGES.deduplicationIdNotice);
        deduplicationId = await this.prompter.input({
          message: MESSAGES.deduplicationIdPrompt,
        });
      }

      choices = await this.prompter.checkbox({
```

```
          message: MESSAGES.messageAttributesPrompt,
          choices: toneChoices,
        });
      }

      await this.snsClient.send(
        new PublishCommand({
          TopicArn: this.topicArn,
          Message: message,
          ...(groupId
            ? {
                MessageGroupId: groupId,
              }
            : {}),
          ...(deduplicationId
            ? {
                MessageDeduplicationId: deduplicationId,
              }
            : {}),
          ...(choices
            ? {
                MessageAttributes: {
                  tone: {
                    DataType: "String.Array",
                    StringValue: JSON.stringify(choices),
                  },
                },
              }
            : {}),
        }),
      );

      const publishAnother = await this.prompter.confirm({
        message: MESSAGES.publishAnother,
      });

      if (publishAnother) {
        await this.publishMessages();
      }
    }

    async receiveAndDeleteMessages() {
      for (const queue of this.queues) {
        const { Messages } = await this.sqsClient.send(
```

```
        new ReceiveMessageCommand({
          QueueUrl: queue.queueUrl,
        }),
      );

      if (Messages) {
        await this.logger.log(
          MESSAGES.messagesReceivedNotice.replace(
            "${QUEUE_NAME}",
            queue.queueName,
          ),
        );
        console.log(Messages);

        await this.sqsClient.send(
          new DeleteMessageBatchCommand({
            QueueUrl: queue.queueUrl,
            Entries: Messages.map((message) => ({
              Id: message.MessageId,
              ReceiptHandle: message.ReceiptHandle,
            })),
          }),
        );
      } else {
        await this.logger.log(
          MESSAGES.noMessagesReceivedNotice.replace(
            "${QUEUE_NAME}",
            queue.queueName,
          ),
        );
      }
    }

    const deleteAndPoll = await this.prompter.confirm({
      message: MESSAGES.deleteAndPollConfirmation,
    });

    if (deleteAndPoll) {
      await this.receiveAndDeleteMessages();
    }
  }

  async destroyResources() {
    for (const subscriptionArn of this.subscriptionArns) {
```

```
      await this.snsClient.send(
        new UnsubscribeCommand({ SubscriptionArn: subscriptionArn }),
      );
    }

    for (const queue of this.queues) {
      await this.sqsClient.send(
        new DeleteQueueCommand({ QueueUrl: queue.queueUrl }),
      );
    }

    if (this.topicArn) {
      await this.snsClient.send(
        new DeleteTopicCommand({ TopicArn: this.topicArn }),
      );
    }
  }

  async start() {
    console.clear();

    try {
      this.logger.logSeparator(MESSAGES.headerWelcome);
      await this.welcome();
      this.logger.logSeparator(MESSAGES.headerFifo);
      await this.confirmFifo();
      this.logger.logSeparator(MESSAGES.headerCreateTopic);
      await this.createTopic();
      this.logger.logSeparator(MESSAGES.headerCreateQueues);
      await this.createQueues();
      this.logger.logSeparator(MESSAGES.headerAttachPolicy);
      await this.attachQueueIamPolicies();
      this.logger.logSeparator(MESSAGES.headerSubscribeQueues);
      await this.subscribeQueuesToTopic();
      this.logger.logSeparator(MESSAGES.headerPublishMessage);
      await this.publishMessages();
      this.logger.logSeparator(MESSAGES.headerReceiveMessages);
      await this.receiveAndDeleteMessages();
    } catch (err) {
      console.error(err);
    } finally {
      await this.destroyResources();
    }
  }
```

```
}
```

- 有关 API 详细信息，请参阅《Amazon SDK for JavaScript API 参考》中的以下主题。
  - CreateQueue
  - CreateTopic
  - DeleteMessageBatch
  - DeleteQueue
  - DeleteTopic
  - GetQueueAttributes
  - Publish
  - ReceiveMessage
  - SetQueueAttributes
  - 订阅
  - Unsubscribe

使用 JavaScript (v3) 软件开发工具包的 Step Functions 示例

以下代码示例向您展示了如何使用带有 Step Functions 的 Amazon SDK for JavaScript (v3) 来执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景 是展示如何通过在同一服务中调用多个函数来完成特定任务的代码示例。

每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关如何在上下文中设置和运行代码的说明。

主题

- 操作

操作

启动状态机运行

以下代码示例展示了如何启动 Step Function 状态机运行。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

import { SFNClient, StartExecutionCommand } from "@aws-sdk/client-sfn";

/**
 * @param {{ sfnClient: SFNClient, stateMachineArn: string }} config
 */
export async function startExecution({ sfnClient, stateMachineArn }) {
  const response = await sfnClient.send(
    new StartExecutionCommand({
      stateMachineArn,
    }),
  );
  console.log(response);
  // Example response:
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '202a9309-c16a-454b-adeb-c4d19afe3bf2',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   executionArn: 'arn:aws:states:us-
east-1:000000000000:execution:MyStateMachine:aaaaaaaa-f787-49fb-a20c-1b61c64eafe6',
  //   startDate: 2024-01-04T15:54:08.362Z
  // }
```

```
  return response;
}

// Call function if run directly
import { fileURLToPath } from "url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  startExecution({ sfnClient: new SFNClient({}), stateMachineArn: "ARN" });
}
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考StartExecution中的。

Amazon STS 使用适用于 JavaScript (v3) 的 SDK 的示例

以下代码示例向您展示了如何通过使用 Amazon SDK for JavaScript (v3) 来执行操作和实现常见场景 Amazon STS。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景 是展示如何通过在同一服务中调用多个函数来完成特定任务的代码示例。

每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关如何在上下文中设置和运行代码的说明。

主题

- 操作

操作

代入角色

以下代码示例说明如何使用代入角色 Amazon STS。

适用于 JavaScript (v3) 的软件开发工具包

Note

还有更多相关信息 GitHub。查找完整示例，了解如何在 Amazon 代码示例存储库中进行设置和运行。

创建客户端。

```
import { STSClient } from "@aws-sdk/client-sts";
// Set the AWS Region.
const REGION = "us-east-1";
// Create an AWS STS service client object.
export const client = new STSClient({ region: REGION });
```

代入 IAM 角色。

```
import { AssumeRoleCommand } from "@aws-sdk/client-sts";

import { client } from "../libs/client.js";

export const main = async () => {
  try {
    // Returns a set of temporary security credentials that you can use to
    // access Amazon Web Services resources that you might not normally
    // have access to.
    const command = new AssumeRoleCommand({
      // The Amazon Resource Name (ARN) of the role to assume.
      RoleArn: "ROLE_ARN",
      // An identifier for the assumed role session.
      RoleSessionName: "session1",
      // The duration, in seconds, of the role session. The value specified
      // can range from 900 seconds (15 minutes) up to the maximum session
      // duration set for the role.
      DurationSeconds: 900,
    });
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考AssumeRole中的。

适用于 JavaScript (v2) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
// Load the AWS SDK for Node.js
const AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

var roleToAssume = {
  RoleArn: "arn:aws:iam::123456789012:role/RoleName",
  RoleSessionName: "session1",
  DurationSeconds: 900,
};
var roleCreds;

// Create the STS service object
var sts = new AWS.STS({ apiVersion: "2011-06-15" });

//Assume Role
sts.assumeRole(roleToAssume, function (err, data) {
  if (err) console.log(err, err.stack);
  else {
    roleCreds = {
      accessKeyId: data.Credentials.AccessKeyId,
      secretAccessKey: data.Credentials.SecretAccessKey,
      sessionToken: data.Credentials.SessionToken,
    };
    stsGetCallerIdentity(roleCreds);
  }
});

//Get Arn of current identity
function stsGetCallerIdentity(creds) {
  var stsParams = { credentials: creds };
  // Create STS service object
  var sts = new AWS.STS(stsParams);
```

```
  sts.getCallerIdentity({}, function (err, data) {
    if (err) {
      console.log(err, err.stack);
    } else {
      console.log(data.Arn);
    }
  });
}
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考AssumeRole中的。

Amazon Web Services Support 使用适用于 JavaScript (v3) 的 SDK 的示例

以下代码示例向您展示了如何通过使用 Amazon SDK for JavaScript (v3) 来执行操作和实现常见场景 Amazon Web Services Support。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景 是展示如何通过在同一服务中调用多个函数来完成特定任务的代码示例。

每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关如何在上下文中设置和运行代码的说明。

开始使用

你好 Amazon Web Services Support

以下代码示例展示了如何开始使用 Amazon Web Services Support。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

调用 `main()` 运行该示例。

```
import {
  DescribeServicesCommand,
  SupportClient,
} from "@aws-sdk/client-support";

// Change the value of 'region' to your preferred AWS Region.
const client = new SupportClient({ region: "us-east-1" });

const getServiceCount = async () => {
  try {
    const { services } = await client.send(new DescribeServicesCommand({}));
    return services.length;
  } catch (err) {
    if (err.name === "SubscriptionRequiredException") {
      throw new Error(
        "You must be subscribed to the AWS Support plan to use this feature.",
      );
    } else {
      throw err;
    }
  }
};

export const main = async () => {
  try {
    const count = await getServiceCount();
    console.log(`Hello, AWS Support! There are ${count} services available.`);
  } catch (err) {
    console.error("Failed to get service count: ", err.message);
  }
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DescribeServices中的。

主题

- 操作
- 场景

操作

向案例添加通信

以下代码示例显示了如何向支持案例添加带有附件的 Amazon Web Services Support 通信。

适用于 JavaScript (v3) 的软件开发工具包

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { AddCommunicationToCaseCommand } from "@aws-sdk/client-support";

import { client } from "../libs/client.js";

export const main = async () => {
  let attachmentSetId;

  try {
    // Add a communication to a case.
    const response = await client.send(
      new AddCommunicationToCaseCommand({
        communicationBody: "Adding an attachment.",
        // Set value to an existing support case id.
        caseId: "CASE_ID",
        // Optional. Set value to an existing attachment set id to add attachments
 to the case.
        attachmentSetId,
      }),
    );
    console.log(response);
    return response;
  } catch (err) {
    console.error(err);
  }
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考AddCommunicationToCase中的。

向集合添加附件

以下代码示例说明如何向 Amazon Web Services Support 附件集添加附件。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { AddAttachmentsToSetCommand } from "@aws-sdk/client-support";

import { client } from "../libs/client.js";

export const main = async () => {
  try {
    // Create a new attachment set or add attachments to an existing set.
    // Provide an 'attachmentSetId' value to add attachments to an existing set.
    // Use AddCommunicationToCase or CreateCase to associate an attachment set with
 a support case.
    const response = await client.send(
      new AddAttachmentsToSetCommand({
        // You can add up to three attachments per set. The size limit is 5 MB per
 attachment.
        attachments: [
          {
            fileName: "example.txt",
            data: new TextEncoder().encode("some example text"),
          },
        ],
      }),
    );
    // Use this ID in AddCommunicationToCase or CreateCase.
    console.log(response.attachmentSetId);
    return response;
  } catch (err) {
    console.error(err);
```

```
  }
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考AddAttachmentsToSet中的。

创建案例

以下代码示例显示了如何创建新 Amazon Web Services Support 案例。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { CreateCaseCommand } from "@aws-sdk/client-support";

import { client } from "../libs/client.js";

export const main = async () => {
  try {
    // Create a new case and log the case id.
    // Important: This creates a real support case in your account.
    const response = await client.send(
      new CreateCaseCommand({
        // The subject line of the case.
        subject: "IGNORE: Test case",
        // Use DescribeServices to find available service codes for each service.
        serviceCode: "service-quicksight-end-user",
        // Use DescribeSecurityLevels to find available severity codes for your
 support plan.
        severityCode: "low",
        // Use DescribeServices to find available category codes for each service.
        categoryCode: "end-user-support",
        // The main description of the support case.
        communicationBody: "This is a test. Please ignore.",
      }),
```

```
    );
    console.log(response.caseId);
    return response;
  } catch (err) {
    console.error(err);
  }
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考CreateCase中的。

描述附件

以下代码示例展示了如何描述 Amazon Web Services Support 案例的附件。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { DescribeAttachmentCommand } from "@aws-sdk/client-support";

import { client } from "../libs/client.js";

export const main = async () => {
  try {
    // Get the metadata and content of an attachment.
    const response = await client.send(
      new DescribeAttachmentCommand({
        // Set value to an existing attachment id.
        // Use DescribeCommunications or DescribeCases to find an attachment id.
        attachmentId: "ATTACHMENT_ID",
      }),
    );
    console.log(response.attachment?.fileName);
    return response;
  } catch (err) {
    console.error(err);
  }
```

```
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DescribeAttachment中的。

描述案例

以下代码示例显示了如何描述 Amazon Web Services Support 案例。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { DescribeCasesCommand } from "@aws-sdk/client-support";

import { client } from "../libs/client.js";

export const main = async () => {
  try {
    // Get all of the unresolved cases in your account.
    // Filter or expand results by providing parameters to the DescribeCasesCommand.
 Refer
    // to the TypeScript definition and the API doc for more information on possible
 parameters.
    // https://docs.aws.amazon.com/AWSJavaScriptSDK/v3/latest/clients/client-
support/interfaces/describecasescommandinput.html
    const response = await client.send(new DescribeCasesCommand({}));
    const caseIds = response.cases.map((supportCase) => supportCase.caseId);
    console.log(caseIds);
    return response;
  } catch (err) {
    console.error(err);
  }
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DescribeCases中的。

描述通信

以下代码示例显示了如何描述案例的 Amazon Web Services Support 通信。

适用于 JavaScript (v3) 的软件开发工具包

Note

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { DescribeCommunicationsCommand } from "@aws-sdk/client-support";

import { client } from "../libs/client.js";

export const main = async () => {
  try {
    // Get all communications for the support case.
    // Filter results by providing parameters to the DescribeCommunicationsCommand.
 Refer
    // to the TypeScript definition and the API doc for more information on possible
 parameters.
    // https://docs.aws.amazon.com/AWSJavaScriptSDK/v3/latest/clients/client-
support/interfaces/describecommunicationscommandinput.html
    const response = await client.send(
      new DescribeCommunicationsCommand({
        // Set value to an existing case id.
        caseId: "CASE_ID",
      }),
    );
    const text = response.communications.map((item) => item.body).join("\n");
    console.log(text);
    return response;
  } catch (err) {
    console.error(err);
  }
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DescribeCommunications中的。

描述严重性级别

以下代码示例显示了如何描述 Amazon Web Services Support 严重性级别。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { DescribeSeverityLevelsCommand } from "@aws-sdk/client-support";

import { client } from "../libs/client.js";

export const main = async () => {
  try {
    // Get the list of severity levels.
    // The available values depend on the support plan for the account.
    const response = await client.send(new DescribeSeverityLevelsCommand({}));
    console.log(response.severityLevels);
    return response;
  } catch (err) {
    console.error(err);
  }
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DescribeSeverityLevels中的。

解析案例

以下代码示例显示了如何解决 Amazon Web Services Support 案例。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import { ResolveCaseCommand } from "@aws-sdk/client-support";

import { client } from "../libs/client.js";

const main = async () => {
  try {
    const response = await client.send(
      new ResolveCaseCommand({
        caseId: "CASE_ID",
      }),
    );

    console.log(response.finalCaseStatus);
    return response;
  } catch (err) {
    console.error(err);
  }
};
```

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考ResolveCase中的。

场景

开始应用场景

以下代码示例演示了操作流程：

- 获取并显示案例的可用服务和严重级别。
- 使用选定的服务、类别和严重性级别创建支持案例。
- 获取并显示当天打开案例的列表。
- 向新案例添加附件集和通信。

- 描述该案例的新附件和通信。
- 解析案例。
- 获取并显示当天未解决的案例列表。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

在终端中运行交互式场景。

```
import {
  AddAttachmentsToSetCommand,
  AddCommunicationToCaseCommand,
  CreateCaseCommand,
  DescribeAttachmentCommand,
  DescribeCasesCommand,
  DescribeCommunicationsCommand,
  DescribeServicesCommand,
  DescribeSeverityLevelsCommand,
  ResolveCaseCommand,
  SupportClient,
} from "@aws-sdk/client-support";
import inquirer from "inquirer";

// Retry an asynchronous function on failure.
const retry = async ({ intervalInMs = 500, maxRetries = 10 }, fn) => {
  try {
    return await fn();
  } catch (err) {
    console.log(`Function call failed. Retrying.`);
    console.error(err.message);
    if (maxRetries === 0) throw err;
    await new Promise((resolve) => setTimeout(resolve, intervalInMs));
    return retry({ intervalInMs, maxRetries: maxRetries - 1 }, fn);
  }
};
```

```
const wrapText = (text, char = "=") => {
  const rule = char.repeat(80);
  return `${rule}\n    ${text}\n${rule}\n`;
};

const client = new SupportClient({ region: "us-east-1" });

// Verify that the account has a Support plan.
export const verifyAccount = async () => {
  const command = new DescribeServicesCommand({});

  try {
    await client.send(command);
  } catch (err) {
    if (err.name === "SubscriptionRequiredException") {
      throw new Error(
        "You must be subscribed to the AWS Support plan to use this feature."
      );
    } else {
      throw err;
    }
  }
};

// Get the list of available services.
export const getService = async () => {
  const { services } = await client.send(new DescribeServicesCommand({}));
  const { selectedService } = await inquirer.prompt({
    name: "selectedService",
    type: "list",
    message:
      "Select a service. Your support case will be created for this service. The
 list of services is truncated for readability.",
    choices: services.slice(0, 10).map((s) => ({ name: s.name, value: s })),
  });
  return selectedService;
};

// Get the list of available support case categories for a service.
export const getCategory = async (service) => {
  const { selectedCategory } = await inquirer.prompt({
    name: "selectedCategory",
    type: "list",
    message: "Select a category.",
```

```
    choices: service.categories.map((c) => ({ name: c.name, value: c })),
  });
  return selectedCategory;
};

// Get the available severity levels for the account.
export const getSeverityLevel = async () => {
  const command = new DescribeSeverityLevelsCommand({});
  const { severityLevels } = await client.send(command);
  const { selectedSeverityLevel } = await inquirer.prompt({
    name: "selectedSeverityLevel",
    type: "list",
    message: "Select a severity level.",
    choices: severityLevels.map((s) => ({ name: s.name, value: s })),
  });
  return selectedSeverityLevel;
};

// Create a new support case and return the caseId.
export const createCase = async ({
  selectedService,
  selectedCategory,
  selectedSeverityLevel,
}) => {
  const command = new CreateCaseCommand({
    subject: "IGNORE: Test case",
    communicationBody: "This is a test. Please ignore.",
    serviceCode: selectedService.code,
    categoryCode: selectedCategory.code,
    severityCode: selectedSeverityLevel.code,
  });
  const { caseId } = await client.send(command);
  return caseId;
};

// Get a list of open support cases created today.
export const getTodaysOpenCases = async () => {
  const d = new Date();
  const startOfToday = new Date(d.getFullYear(), d.getMonth(), d.getDate());
  const command = new DescribeCasesCommand({
    includeCommunications: false,
    afterTime: startOfToday.toISOString(),
  });
```

```
  const { cases } = await client.send(command);

  if (cases.length === 0) {
    throw new Error(
      "Unexpected number of cases. Expected more than 0 open cases."
    );
  }
  return cases;
};

// Create an attachment set.
export const createAttachmentSet = async () => {
  const command = new AddAttachmentsToSetCommand({
    attachments: [
      {
        fileName: "example.txt",
        data: new TextEncoder().encode("some example text"),
      },
    ],
  });
  const { attachmentSetId } = await client.send(command);
  return attachmentSetId;
};

export const linkAttachmentSetToCase = async (attachmentSetId, caseId) => {
  const command = new AddCommunicationToCaseCommand({
    attachmentSetId,
    caseId,
    communicationBody: "Adding attachment set to case.",
  });
  await client.send(command);
};

// Get all communications for a support case.
export const getCommunications = async (caseId) => {
  const command = new DescribeCommunicationsCommand({
    caseId,
  });
  const { communications } = await client.send(command);
  return communications;
};

// Get an attachment set.
export const getFirstAttachment = (communications) => {
```

```
  const firstCommWithAttachment = communications.find(
    (c) => c.attachmentSet.length > 0
  );
  return firstCommWithAttachment?.attachmentSet[0].attachmentId;
};

// Get an attachment.
export const getAttachment = async (attachmentId) => {
  const command = new DescribeAttachmentCommand({
    attachmentId,
  });
  const { attachment } = await client.send(command);
  return attachment;
};

// Resolve the case matching the given case ID.
export const resolveCase = async (caseId) => {
  const { shouldResolve } = await inquirer.prompt({
    name: "shouldResolve",
    type: "confirm",
    message: `Do you want to resolve ${caseId}?`,
  });

  if (shouldResolve) {
    const command = new ResolveCaseCommand({
      caseId: caseId,
    });

    await client.send(command);
    return true;
  }
  return false;
};

// Find a specific case in the list of provided cases by case ID.
// If the case is not found, and the results are paginated, continue
// paging through the results.
export const findCase = async ({ caseId, cases, nextToken }) => {
  const foundCase = cases.find((c) => c.caseId === caseId);

  if (foundCase) {
    return foundCase;
  }
```

```
  if (nextToken) {
    const response = await client.send(
      new DescribeCasesCommand({
        nextToken,
        includeResolvedCases: true,
      })
    );
    return findCase({
      caseId,
      cases: response.cases,
      nextToken: response.nextToken,
    });
  }

  throw new Error(`${caseId} not found.`);
};

// Get all cases created today.
export const getTodaysResolvedCases = async (caseIdToWaitFor) => {
  const d = new Date("2023-01-18");
  const startOfToday = new Date(d.getFullYear(), d.getMonth(), d.getDate());
  const command = new DescribeCasesCommand({
    includeCommunications: false,
    afterTime: startOfToday.toISOString(),
    includeResolvedCases: true,
  });
  const { cases, nextToken } = await client.send(command);
  await findCase({ cases, caseId: caseIdToWaitFor, nextToken });
  return cases.filter((c) => c.status === "resolved");
};

const main = async () => {
  let caseId;
  try {
    console.log(wrapText("Welcome to the AWS Support basic usage scenario."));

    // Verify that the account is subscribed to support.
    await verifyAccount();

    // Provided a truncated list of services and prompt the user to select one.
    const selectedService = await getService();

    // Provided the categories for the selected service and prompt the user to
 select one.
```

```
    const selectedCategory = await getCategory(selectedService);

    // Provide the severity available severity levels for the account and prompt the
 user to select one.
    const selectedSeverityLevel = await getSeverityLevel();

    // Create a support case.
    console.log("\nCreating a support case.");
    caseId = await createCase({
      selectedService,
      selectedCategory,
      selectedSeverityLevel,
    });
    console.log(`Support case created: ${caseId}`);

    // Display a list of open support cases created today.
    const todaysOpenCases = await retry(
      { intervalInMs: 1000, maxRetries: 15 },
      getTodaysOpenCases
    );
    console.log(
      `\nOpen support cases created today: ${todaysOpenCases.length}`
    );
    console.log(todaysOpenCases.map((c) => `${c.caseId}`).join("\n"));

    // Create an attachment set.
    console.log("\nCreating an attachment set.");
    const attachmentSetId = await createAttachmentSet();
    console.log(`Attachment set created: ${attachmentSetId}`);

    // Add the attachment set to the support case.
    console.log(`\nAdding attachment set to ${caseId}`);
    await linkAttachmentSetToCase(attachmentSetId, caseId);
    console.log(`Attachment set added to ${caseId}`);

    // List the communications for a support case.
    console.log(`\nListing communications for ${caseId}`);
    const communications = await getCommunications(caseId);
    console.log(
      communications
        .map(
          (c) =>
            `Communication created on ${c.timeCreated}. Has
 ${c.attachmentSet.length} attachments.`
```

```
        )
        .join("\n")
    );

    // Describe the first attachment.
    console.log(`\nDescribing attachment ${attachmentSetId}`);
    const attachmentId = getFirstAttachment(communications);
    const attachment = await getAttachment(attachmentId);
    console.log(
      `Attachment is the file '${
        attachment.fileName
      }' with data: \n${new TextDecoder().decode(attachment.data)}`
    );

    // Confirm that the support case should be resolved.
    const isResolved = await resolveCase(caseId);
    if (isResolved) {
      // List the resolved cases and include the one previously created.
      // Resolved cases can take a while to appear.
      console.log(
        "\nWaiting for case status to be marked as resolved. This can take some
 time."
      );
      const resolvedCases = await retry(
        { intervalInMs: 20000, maxRetries: 15 },
        () => getTodaysResolvedCases(caseId)
      );
      console.log("Resolved cases:");
      console.log(resolvedCases.map((c) => c.caseId).join("\n"));
    }
  } catch (err) {
    console.error(err);
  }
};
```

- 有关 API 详细信息，请参阅《Amazon SDK for JavaScript API 参考》中的以下主题。
  - AddAttachmentsToSet
  - AddCommunicationToCase
  - CreateCase
  - DescribeAttachment

- DescribeCases
- DescribeCommunications
- DescribeServices
- DescribeSeverityLevels
- ResolveCase

使用适用于 JavaScript (v3) 的软件开发工具包的 Amazon Transcribe 示例

以下代码示例向您展示了如何使用带有 Amazon Transcribe 的 Amazon SDK for JavaScript (v3) 来执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景 是展示如何通过在同一服务中调用多个函数来完成特定任务的代码示例。

每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关如何在上下文中设置和运行代码的说明。

主题

- 操作

操作

删除医疗转录作业

以下代码示例展示了如何删除 Amazon Transcribe Medical 转录任务。

适用于 JavaScript (v3) 的软件开发工具包

Note

还有更多相关信息 GitHub。查找完整示例，了解如何在 Amazon 代码示例存储库中进行设置和运行。

创建客户端。

```
const { TranscribeClient } = require("@aws-sdk/client-transcribe");
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon Transcribe service client object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

删除医疗转录作业。

```
// Import the required AWS SDK clients and commands for Node.js
import { DeleteMedicalTranscriptionJobCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "./libs/transcribeClient.js";

// Set the parameters
export const params = {
  MedicalTranscriptionJobName: "MEDICAL_JOB_NAME", // For example,
 'medical_transciption_demo'
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new DeleteMedicalTranscriptionJobCommand(params)
    );
    console.log("Success - deleted");
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DeleteMedicalTranscriptionJob中的。

删除转录任务

以下代码示例展示了如何删除 Amazon Transcribe 转录任务。

适用于 JavaScript (v3) 的软件开发工具包

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

删除转录作业。

```
// Import the required AWS SDK clients and commands for Node.js
import { DeleteTranscriptionJobCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "./libs/transcribeClient.js";

// Set the parameters
export const params = {
  TranscriptionJobName: "JOB_NAME", // Required. For example, 'transciption_demo'
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new DeleteTranscriptionJobCommand(params)
    );
    console.log("Success - deleted");
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

创建客户端。

```
const { TranscribeClient } = require("@aws-sdk/client-transcribe");
// Set the AWS Region.
```

```
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon Transcribe service client object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考DeleteTranscriptionJob中的。

列出医疗转录任务

以下代码示例展示了如何列出 Amazon Transcribe Medical 转录任务。

适用于 JavaScript (v3) 的软件开发工具包

> Note
>
> 还有更多相关信息 GitHub。查找完整示例，了解如何在 Amazon 代码示例存储库中进行设置和运行。

创建客户端。

```
const { TranscribeClient } = require("@aws-sdk/client-transcribe");
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon Transcribe service client object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

列出医疗转录作业。

```
// Import the required AWS SDK clients and commands for Node.js
import { StartMedicalTranscriptionJobCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "./libs/transcribeClient.js";

// Set the parameters
export const params = {
```

```
  MedicalTranscriptionJobName: "MEDICAL_JOB_NAME", // Required
  OutputBucketName: "OUTPUT_BUCKET_NAME", // Required
  Specialty: "PRIMARYCARE", // Required. Possible values are 'PRIMARYCARE'
  Type: "JOB_TYPE", // Required. Possible values are 'CONVERSATION' and 'DICTATION'
  LanguageCode: "LANGUAGE_CODE", // For example, 'en-US'
  MediaFormat: "SOURCE_FILE_FORMAT", // For example, 'wav'
  Media: {
    MediaFileUri: "SOURCE_FILE_LOCATION",
    // The S3 object location of the input media file. The URI must be in the same
 region
    // as the API endpoint that you are calling.For example,
    // "https://transcribe-demo.s3-REGION.amazonaws.com/hello_world.wav"
  },
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new StartMedicalTranscriptionJobCommand(params)
    );
    console.log("Success - put", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考ListMedicalTranscriptionJobs中的。

列出转录任务

以下代码示例展示了如何列出 Amazon Transcribe 转录任务。

适用于 JavaScript (v3) 的软件开发工具包

Note

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

列出转录作业。

```
// Import the required AWS SDK clients and commands for Node.js

import { ListTranscriptionJobsCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "./libs/transcribeClient.js";

// Set the parameters
export const params = {
  JobNameContains: "KEYWORD", // Not required. Returns only transcription
  // job names containing this string
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new ListTranscriptionJobsCommand(params)
    );
    console.log("Success", data.TranscriptionJobSummaries);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

创建客户端。

```
const { TranscribeClient } = require("@aws-sdk/client-transcribe");
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon Transcribe service client object.
const transcribeClient = new TranscribeClient({ region: REGION });
```

```
export { transcribeClient };
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考ListTranscriptionJobs中的。

启动医疗转录任务

以下代码示例展示了如何启动 Amazon Transcribe Medical 转录任务。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。查找完整示例，了解如何在 Amazon 代码示例存储库中进行设置和运行。

创建客户端。

```
const { TranscribeClient } = require("@aws-sdk/client-transcribe");
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon Transcribe service client object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

启动医疗转录作业。

```
// Import the required AWS SDK clients and commands for Node.js
import { StartMedicalTranscriptionJobCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "./libs/transcribeClient.js";

// Set the parameters
export const params = {
  MedicalTranscriptionJobName: "MEDICAL_JOB_NAME", // Required
  OutputBucketName: "OUTPUT_BUCKET_NAME", // Required
  Specialty: "PRIMARYCARE", // Required. Possible values are 'PRIMARYCARE'
  Type: "JOB_TYPE", // Required. Possible values are 'CONVERSATION' and 'DICTATION'
  LanguageCode: "LANGUAGE_CODE", // For example, 'en-US'
```

```
  MediaFormat: "SOURCE_FILE_FORMAT", // For example, 'wav'
  Media: {
    MediaFileUri: "SOURCE_FILE_LOCATION",
    // The S3 object location of the input media file. The URI must be in the same
 region
    // as the API endpoint that you are calling.For example,
    // "https://transcribe-demo.s3-REGION.amazonaws.com/hello_world.wav"
  },
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new StartMedicalTranscriptionJobCommand(params)
    );
    console.log("Success - put", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。
- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考StartMedicalTranscriptionJob中的。

启动转录任务

以下代码示例展示了如何启动 Amazon Transcribe 转录任务。

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

启动转录作业。

```
// Import the required AWS SDK clients and commands for Node.js
import { StartTranscriptionJobCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "./libs/transcribeClient.js";

// Set the parameters
export const params = {
  TranscriptionJobName: "JOB_NAME",
  LanguageCode: "LANGUAGE_CODE", // For example, 'en-US'
  MediaFormat: "SOURCE_FILE_FORMAT", // For example, 'wav'
  Media: {
    MediaFileUri: "SOURCE_LOCATION",
    // For example, "https://transcribe-demo.s3-REGION.amazonaws.com/
hello_world.wav"
  },
  OutputBucketName: "OUTPUT_BUCKET_NAME"
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new StartTranscriptionJobCommand(params)
    );
    console.log("Success - put", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

创建客户端。

```
const { TranscribeClient } = require("@aws-sdk/client-transcribe");
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon Transcribe service client object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

- 有关更多信息，请参阅 Amazon SDK for JavaScript 开发人员指南。

- 有关 API 的详细信息，请参阅 Amazon SDK for JavaScript API 参考StartTranscriptionJob中的。

使用适用于 JavaScript (v3) 的 SDK 的跨服务示例

以下示例应用程序使用 Amazon SDK for JavaScript (v3) 跨多个应用程序运行 Amazon Web Services。

跨服务示例以高级体验为目标，以便您开始构建应用程序。

示例

- 构建 Amazon Transcribe 应用程序
- 构建 Amazon Transcribe 流式传输应用程序
- 构建应用程序以将数据提交到 DynamoDB 表
- 创建 Amazon Lex 聊天机器人来吸引网站访问者
- 创建照片资产管理应用程序，让用户能够使用标签管理照片
- 创建 Web 应用程序来跟踪 DynamoDB 数据
- 创建 Aurora Serverless 工作项跟踪器
- 创建 Amazon Textract 浏览器应用程序
- 创建用于分析客户反馈和合成音频的应用程序
- 使用软件开发工具包使用 Amazon Rekognition 检测图像中的个人防护装备 Amazon
- 使用软件开发工具包使用 Amazon Rekognition 检测图像中的物体 Amazon
- 使用 Amazon Rekognition 使用软件开发工具包检测视频中的人物和物体 Amazon
- 从浏览器调用 Lambda 函数
- 使用 API Gateway 调用 Lambda 函数
- 使用 Step Functions 调用 Lambda 函数
- 使用计划的事件调用 Lambda 函数

构建 Amazon Transcribe 应用程序

适用于 JavaScript (v3) 的软件开发工具包

创建一个使用 Amazon Transcribe 在浏览器中转录和显示录音的应用程序。该应用程序使用两个 Amazon Simple Storage Service（Amazon S3）桶，一个用于托管应用程序代码，另一个用于存

储转录。该应用程序使用 Amazon Cognito 用户池对您的用户进行身份验证。经过身份验证的用户拥有 Amazon Identity and Access Management (IAM) 访问所需 Amazon 服务的权限。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例GitHub。

该示例也可在 Amazon SDK for JavaScript v3 开发人员指南中找到。

本示例中使用的服务

- Amazon Cognito Identity
- Amazon S3
- Amazon Transcribe

构建 Amazon Transcribe 流式传输应用程序

适用于 JavaScript (v3) 的软件开发工具包

演示了如何使用 Amazon Transcribe 构建可实时录制、转录与翻译实时音频，并通过 Amazon Simple Email Service (Amazon SES) 以电子邮件发送结果的应用程序。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例GitHub。

本示例中使用的服务

- Amazon Comprehend
- Amazon SES
- Amazon Transcribe
- Amazon Translate

构建应用程序以将数据提交到 DynamoDB 表

适用于 JavaScript (v3) 的软件开发工具包

此示例展示了如何构建一个应用程序，使用户能够向 Amazon DynamoDB 表提交数据，并使用 Amazon Simple Notification Service (Amazon SNS) 向管理员发送文本消息。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例GitHub。

该示例也可在 Amazon SDK for JavaScript v3 开发人员指南中找到。

本示例中使用的服务

- DynamoDB
- Amazon SNS

创建 Amazon Lex 聊天机器人来吸引网站访问者

适用于 JavaScript (v3) 的软件开发工具包

展示如何使用 Amazon Lex API 在 Web 应用程序中创建聊天机器人，以吸引您的网站访问者。

有关如何设置和运行的完整源代码和说明，请参阅 Amazon SDK for JavaScript 开发者指南中的构建 Amazon Lex 聊天机器人的完整示例。

本示例中使用的服务

- Amazon Comprehend
- Amazon Lex
- Amazon Translate

创建照片资产管理应用程序，让用户能够使用标签管理照片

适用于 JavaScript (v3) 的软件开发工具包

演示了如何开发照片资产管理应用程序，该应用程序使用 Amazon Rekognition 检测图像中的标签并将其存储以供日后检索。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例 GitHub。

要深入了解这个例子的起源，请参阅 Amazon 社区上的博文。

本示例中使用的服务

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3

- Amazon SNS

创建 Web 应用程序来跟踪 DynamoDB 数据

适用于 JavaScript (v3) 的软件开发工具包

演示了如何使用 Amazon DynamoDB API 创建用于跟踪 DynamoDB 工作数据的动态 Web 应用程序。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例GitHub。

本示例中使用的服务

- DynamoDB
- Amazon SES

创建 Aurora Serverless 工作项跟踪器

适用于 JavaScript (v3) 的软件开发工具包

演示如何使用 Amazon SDK for JavaScript (v3) 创建一个 Web 应用程序，该应用程序使用亚马逊简单电子邮件服务 (Amazon SES) Service 跟踪亚马逊 Aurora 数据库中的工作项目并通过电子邮件发送报告。此示例使用由 React.js 构建的前端与 Express Node.js 后端进行交互。

- 将 React.js 网络应用程序与集成 Amazon Web Services。
- 列出、添加以及更新 Aurora 表中的项目。
- 使用 Amazon SES 以电子邮件形式发送已筛选工作项的报告。
- 使用随附的 Amazon CloudFormation 脚本部署和管理示例资源。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例GitHub。

本示例中使用的服务

- Aurora
- Amazon RDS
- Amazon RDS 数据服务
- Amazon SES

创建 Amazon Textract 浏览器应用程序

适用于 JavaScript (v3) 的软件开发工具包

演示如何使用 Amazon SDK for JavaScript 来构建 React 应用程序，该应用程序使用 Amazon Textract 从文档图像中提取数据并将其显示在交互式网页中。此示例在 Web 浏览器中运行，需要经过身份验证的 Amazon Cognito 身份才能获得凭证。它使用 Amazon Simple Storage Service（Amazon S3）进行存储；对于通知，它将轮询订阅 Amazon Simple Notification Service（Amazon SNS）主题的 Amazon Simple Queue Service（Amazon SQS）队列。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例GitHub。

本示例中使用的服务

- Amazon Cognito Identity
- Amazon S3
- Amazon SNS
- Amazon SQS
- Amazon Textract

创建用于分析客户反馈和合成音频的应用程序

适用于 JavaScript (v3) 的软件开发工具包

此示例应用程序可分析并存储客户反馈卡。具体来说，它满足了纽约市一家虚构酒店的需求。酒店以实体意见卡的形式收集来自不同语种的客人的反馈。该反馈通过 Web 客户端上传到应用程序中。意见卡图片上传后，将执行以下步骤：

- 使用 Amazon Textract 从图片中提取文本。
- Amazon Comprehend 确定所提取文本的情绪及其语言。
- 使用 Amazon Translate 将所提取文本翻译为英语。
- Amazon Polly 根据所提取文本合成音频文件。

完整的应用程序可使用 Amazon CDK 进行部署。有关源代码和部署说明，请参阅中的项目GitHub。以下摘录显示了在 Lambda 函数中 Amazon SDK for JavaScript 是如何使用的。

```
import {
  ComprehendClient,
  DetectDominantLanguageCommand,
```

```
  DetectSentimentCommand,
} from "@aws-sdk/client-comprehend";

/**
 * Determine the language and sentiment of the extracted text.
 *
 * @param {{ source_text: string}} extractTextOutput
 */
export const handler = async (extractTextOutput) => {
  const comprehendClient = new ComprehendClient({});

  const detectDominantLanguageCommand = new DetectDominantLanguageCommand({
    Text: extractTextOutput.source_text,
  });

  // The source language is required for sentiment analysis and
  // translation in the next step.
  const { Languages } = await comprehendClient.send(
    detectDominantLanguageCommand,
  );

  const languageCode = Languages[0].LanguageCode;

  const detectSentimentCommand = new DetectSentimentCommand({
    Text: extractTextOutput.source_text,
    LanguageCode: languageCode,
  });

  const { Sentiment } = await comprehendClient.send(detectSentimentCommand);

  return {
    sentiment: Sentiment,
    language_code: languageCode,
  };
};
```

```
import {
  DetectDocumentTextCommand,
  TextractClient,
} from "@aws-sdk/client-textract";

/**
 * Fetch the S3 object from the event and analyze it using Amazon Textract.
```

```
 *
 * @param {import("@types/aws-lambda").EventBridgeEvent<"Object Created">}
 eventBridgeS3Event
 */
export const handler = async (eventBridgeS3Event) => {
  const textractClient = new TextractClient();

  const detectDocumentTextCommand = new DetectDocumentTextCommand({
    Document: {
      S3Object: {
        Bucket: eventBridgeS3Event.bucket,
        Name: eventBridgeS3Event.object,
      },
    },
  });

  // Textract returns a list of blocks. A block can be a line, a page, word, etc.
  // Each block also contains geometry of the detected text.
  // For more information on the Block type, see https://docs.aws.amazon.com/
textract/latest/dg/API_Block.html.
  const { Blocks } = await textractClient.send(detectDocumentTextCommand);

  // For the purpose of this example, we are only interested in words.
  const extractedWords = Blocks.filter((b) => b.BlockType === "WORD").map(
    (b) => b.Text,
  );

  return extractedWords.join(" ");
};
```

```
import { PollyClient, SynthesizeSpeechCommand } from "@aws-sdk/client-polly";
import { S3Client } from "@aws-sdk/client-s3";
import { Upload } from "@aws-sdk/lib-storage";

/**
 * Synthesize an audio file from text.
 *
 * @param {{ bucket: string, translated_text: string, object: string}}
 sourceDestinationConfig
 */
export const handler = async (sourceDestinationConfig) => {
  const pollyClient = new PollyClient({});
```

```
  const synthesizeSpeechCommand = new SynthesizeSpeechCommand({
    Engine: "neural",
    Text: sourceDestinationConfig.translated_text,
    VoiceId: "Ruth",
    OutputFormat: "mp3",
  });

  const { AudioStream } = await pollyClient.send(synthesizeSpeechCommand);

  const audioKey = `${sourceDestinationConfig.object}.mp3`;

  // Store the audio file in S3.
  const s3Client = new S3Client();
  const upload = new Upload({
    client: s3Client,
    params: {
      Bucket: sourceDestinationConfig.bucket,
      Key: audioKey,
      Body: AudioStream,
      ContentType: "audio/mp3",
    },
  });

  await upload.done();
  return audioKey;
};
```

```
import {
  TranslateClient,
  TranslateTextCommand,
} from "@aws-sdk/client-translate";

/**
 * Translate the extracted text to English.
 *
 * @param {{ extracted_text: string, source_language_code: string}}
 textAndSourceLanguage
 */
export const handler = async (textAndSourceLanguage) => {
  const translateClient = new TranslateClient({});

  const translateCommand = new TranslateTextCommand({
    SourceLanguageCode: textAndSourceLanguage.source_language_code,
```

```
    TargetLanguageCode: "en",
    Text: textAndSourceLanguage.extracted_text,
  });

  const { TranslatedText } = await translateClient.send(translateCommand);

  return { translated_text: TranslatedText };
};
```

本示例中使用的服务

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

使用软件开发工具包使用 Amazon Rekognition 检测图像中的个人防护装备 Amazon

适用于 JavaScript (v3) 的软件开发工具包

演示如何使用 Amazon Rekognition Amazon SDK for JavaScript 和，创建用于检测亚马逊简单存储服务 (Amazon S3) 存储桶中图像中的个人防护装备 (PPE) 的应用程序。该应用程序将结果保存到 Amazon DynamoDB 表，然后使用 Amazon Simple Email Service (Amazon SES) 向管理员发送包含结果的电子邮件通知。

了解如何：

- 使用 Amazon Cognito 创建未经身份验证的用户。
- 使用 Amazon Rekognition 分析包含 PPE 的图像。
- 为 Amazon SES 验证电子邮件地址。
- 使用结果更新 DynamoDB 表。
- 使用 Amazon SES 发送电子邮件通知。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例GitHub。

本示例中使用的服务

- DynamoDB

- Amazon Rekognition
- Amazon S3
- Amazon SES

使用软件开发工具包使用 Amazon Rekognition 检测图像中的物体 Amazon

适用于 JavaScript (v3) 的软件开发工具包

演示如何使用 Amazon Rekogn Amazon SDK for JavaScript ition 和，创建一款应用程序，该应用程序使用 Amazon Rekognition 按类别识别位于亚马逊简单存储服务 (Amazon S3) Simple S3 存储桶中的图像中的对象。该应用程序使用 Amazon Simple Email Service (Amazon SES) 向管理员发送包含结果的电子邮件通知。

了解如何：

- 使用 Amazon Cognito 创建未经身份验证的用户。
- 使用 Amazon Rekognition 分析包含对象的图像。
- 为 Amazon SES 验证电子邮件地址。
- 使用 Amazon SES 发送电子邮件通知。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例GitHub。

本示例中使用的服务

- Amazon Rekognition
- Amazon S3
- Amazon SES

使用 Amazon Rekognition 使用软件开发工具包检测视频中的人物和物体 Amazon

适用于 JavaScript (v3) 的软件开发工具包

演示如何使用 Amazon Rekognition Amazon SDK for JavaScript 和，创建用于检测位于亚马逊简单存储服务 (Amazon S3) 存储段中的视频中的人脸和物体的应用程序。该应用程序使用 Amazon Simple Email Service (Amazon SES) 向管理员发送包含结果的电子邮件通知。

了解如何：

- 使用 Amazon Cognito 创建未经身份验证的用户。
- 使用 Amazon Rekognition 分析包含 PPE 的图像。
- 为 Amazon SES 验证电子邮件地址。
- 使用 Amazon SES 发送电子邮件通知。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例GitHub。

本示例中使用的服务

- Amazon Rekognition
- Amazon S3
- Amazon SES

从浏览器调用 Lambda 函数

适用于 JavaScript (v2) 的软件开发工具包

您可以创建一个基于浏览器的应用程序，该应用程序使用 Amazon Lambda 函数更新包含用户选择的 Amazon DynamoDB 表。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例GitHub。

本示例中使用的服务

- DynamoDB
- Lambda

适用于 JavaScript (v3) 的软件开发工具包

您可以创建一个基于浏览器的应用程序，该应用程序使用 Amazon Lambda 函数更新包含用户选择的 Amazon DynamoDB 表。此应用程序使用 Amazon SDK for JavaScript v3。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例GitHub。

本示例中使用的服务

- DynamoDB
- Lambda

使用 API Gateway 调用 Lambda 函数

适用于 JavaScript (v3) 的软件开发工具包

演示如何使用 Lambda JavaScript 运行时 API 创建 Amazon Lambda 函数。此示例调用不同的 Amazon 服务来执行特定的用例。此示例展示了如何创建通过 Amazon API Gateway 调用的 Lambda 函数，该函数扫描 Amazon DynamoDB 表获取工作周年纪念日，并使用 Amazon Simple Notification Service (Amazon SNS)向员工发送文本消息，祝贺他们的周年纪念日。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例GitHub。

该示例也可在 Amazon SDK for JavaScript v3 开发人员指南中找到。

本示例中使用的服务

- API Gateway
- DynamoDB
- Lambda
- Amazon SNS

使用 Step Functions 调用 Lambda 函数

适用于 JavaScript (v3) 的软件开发工具包

演示如何使用 Amazon Step Functions 和创建 Amazon 无服务器工作流程。 Amazon SDK for JavaScript每个工作流程步骤都是使用 Amazon Lambda 函数实现的。

Lambda 是一项计算服务，使您无需预置或管理服务器即可运行代码。Step Functions 是一项无服务器编排服务，可让您搭配使用 Lambda 函数和其他 Amazon 服务来构建业务关键型应用程序。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例GitHub。

该示例也可在 Amazon SDK for JavaScript v3 开发人员指南中找到。

本示例中使用的服务

- DynamoDB
- Lambda
- Amazon SES
- Step Functions

使用计划的事件调用 Lambda 函数

适用于 JavaScript (v3) 的软件开发工具包

演示如何创建调用函数的 Amazon EventBridge 计划事件。 Amazon Lambda 配置 EventBridge 为使用 cron 表达式来调度 Lambda 函数的调用时间。在此示例中，您将使用 Lambda 运行时 API 创建一个 Lambda 函数。 JavaScript 此示例调用不同的 Amazon 服务来执行特定的用例。此示例展示了如何创建一个应用程序，在其一周年纪念日时向员工发送移动短信表示祝贺。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例GitHub。

该示例也可在 Amazon SDK for JavaScript v3 开发人员指南中找到。

本示例中使用的服务

- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

本 Amazon 产品或服务的安全性

云安全性一直是 Amazon Web Services (Amazon) 的重中之重。作为 Amazon 客户，您将从专为满足大多数安全敏感型企业的要求而打造的数据中心和网络架构中受益。安全是双方共同承担 Amazon 的责任。责任共担模式将其描述为云的安全性和云中的安全性。

云安全 — Amazon 负责保护运行 Amazon 云中提供的所有服务的基础架构，并为您提供可以安全使用的服务。我们的安全责任是重中之重 Amazon，作为Amazon 合规计划的一部分，第三方审计师定期测试和验证我们安全的有效性。

云端安全 — 您的责任由您使用的 Amazon 服务以及其他因素决定，包括数据的敏感性、组织的要求以及适用的法律和法规。

本 Amazon 产品或服务通过其支持的特定 Amazon Web Services (Amazon) 服务遵循分担责任模式。有关 Amazon 服务安全信息，请参阅Amazon 服务安全文档页面和合规计划合 Amazon 规工作范围内的Amazon 服务。

主题

- 本 Amazon 产品或服务中的数据保护
- Identity and Access Management
- 此 Amazon 产品或服务的合规性验证
- 本 Amazon 产品或服务的弹性
- 本 Amazon 产品或服务的基础设施安全
- 强制使用最低版本的 TLS

本 Amazon 产品或服务中的数据保护

分 Amazon 分担责任模型适用于本 Amazon 产品或服务中的数据保护。如本模型所述 Amazon ，负责保护运行所有内容的全球基础架构 Amazon Web Services 云。您负责维护对托管在此基础设施上的内容的控制。您还负责您所使用的 Amazon Web Services 的安全配置和管理任务。有关数据隐私的更多信息，请参阅数据隐私常见问题解答。

出于数据保护目的，我们建议您保护 Amazon Web Services 账户 凭证并使用 Amazon IAM Identity Center 或 Amazon Identity and Access Management (IAM) 设置个人用户。这样，每个用户只获得履行其工作职责所需的权限。我们还建议您通过以下方式保护数据：

- 对每个账户使用多重身份验证 (MFA)。

- 使用 SSL/TLS 与资源通信。 Amazon 我们要求使用 TLS 1.2，建议使用 TLS 1.3。
- 使用设置 API 和用户活动日志 Amazon CloudTrail。
- 使用 Amazon 加密解决方案以及其中的所有默认安全控件 Amazon Web Services。
- 使用高级托管安全服务（例如 Amazon Macie），它有助于发现和保护存储在 Amazon S3 中的敏感数据。
- 如果您在 Amazon 通过命令行界面或 API 进行访问时需要经过 FIPS 140-2 验证的加密模块，请使用 FIPS 端点。有关可用的 FIPS 端点的更多信息，请参阅《美国联邦信息处理标准(FIPS) 第 140-2 版》。

我们强烈建议您切勿将机密信息或敏感信息（如您客户的电子邮件地址）放入标签或自由格式文本字段（如名称字段）。这包括您使用控制台、API 或 Amazon SDK Amazon Web Services 使用本 Amazon 产品或服务或其他产品或服务时。 Amazon CLI在用于名称的标签或自由格式文本字段中输入的任何数据都可能会用于计费或诊断日志。如果您向外部服务器提供网址，强烈建议您不要在网址中包含凭证信息来验证对该服务器的请求。

Identity and Access Management

Amazon Identity and Access Management (IAM) Amazon Web Service 可帮助管理员安全地控制对 Amazon 资源的访问权限。IAM 管理员控制谁可以进行身份验证（登录）和授权（拥有权限）使用 Amazon 资源。您可以使用 IAM Amazon Web Service ，无需支付额外费用。

主题

- 受众
- 使用身份进行身份验证
- 使用策略管理访问
- 如何 Amazon Web Services 使用 IAM
- 对 Amazon 身份和访问进行故障排除

受众

您的使用方式 Amazon Identity and Access Management (IAM) 会有所不同，具体取决于您所做的工作 Amazon。

服务用户-如果您 Amazon Web Services 曾经完成工作，则您的管理员会为您提供所需的凭证和权限。当你使用更多 Amazon 功能来完成工作时，你可能需要额外的权限。了解如何管理访问权限有助于您

向管理员请求适合的权限。如果您无法访问中的功能 Amazon，请参阅对 Amazon 身份和访问进行故障排除或 Amazon Web Service 您正在使用的用户指南。

服务管理员-如果您负责公司的 Amazon 资源，则可能拥有完全访问权限 Amazon。您的工作是确定您的服务用户应访问哪些 Amazon 功能和资源。然后，您必须向 IAM 管理员提交请求以更改服务用户的权限。请查看该页面上的信息以了解 IAM 的基本概念。要详细了解您的公司如何使用 IAM Amazon，请参阅 Amazon Web Service 您正在使用的用户指南。

IAM 管理员：如果您是 IAM 管理员，您可能希望了解如何编写策略以管理对 Amazon的访问权限的详细信息。要查看您可以在 IAM 中使用的 Amazon 基于身份的策略示例，请参阅 Amazon Web Service 您正在使用的用户指南。

使用身份进行身份验证

身份验证是您 Amazon 使用身份凭证登录的方式。您必须以 IAM 用户身份或通过担 Amazon Web Services 账户根用户任 IAM 角色进行身份验证（登录 Amazon）。

如果您 Amazon 以编程方式访问，则会 Amazon 提供软件开发套件 (SDK) 和命令行接口 (CLI)，以便使用您的凭据对请求进行加密签名。如果您不使用 Amazon 工具，则必须自己签署请求。有关使用推荐的方法自行签署请求的更多信息，请参阅 IAM 用户指南中的签署 Amazon API 请求。

无论使用何种身份验证方法，您可能需要提供其他安全信息。例如， Amazon 建议您使用多重身份验证 (MFA) 来提高账户的安全性。要了解更多信息，请参阅《IAM 用户指南》中的在 Amazon中使用多重身份验证 (MFA)。

Amazon Web Services 账户 root 用户

创建时 Amazon Web Services 账户，首先要有一个登录身份，该身份可以完全访问账户中的所有资源 Amazon Web Services 和资源。此身份被称为 Amazon Web Services 账户 root 用户，使用您创建账户时使用的电子邮件地址和密码登录即可访问该身份。强烈建议不要使用根用户执行日常任务。保护好根用户凭证，并使用这些凭证来执行仅根用户可以执行的任务。有关要求以根用户身份登录的任务的完整列表，请参阅《IAM 用户指南》中的需要根用户凭证的任务。

联合身份

作为最佳实践，要求人类用户（包括需要管理员访问权限的用户）使用与身份提供商的联合身份验证 Amazon Web Services 通过临时证书进行访问。

联合身份是指您的企业用户目录、Web 身份提供商、Identity C 或者任何使用 Amazon Web Services 通过身份源提供的凭据进行访问的用户。 Amazon Directory Service当联合身份访问时 Amazon Web Services 账户，他们将扮演角色，角色提供临时证书。

IAM 用户和群组

I AM 用户是您 Amazon Web Services 账户 内部对个人或应用程序具有特定权限的身份。在可能的情况下，建议使用临时凭证，而不是创建具有长期凭证（如密码和访问密钥）的 IAM 用户。但是，如果有一些特定的使用场景需要长期凭证以及 IAM 用户，我们建议轮换访问密钥。有关更多信息，请参阅《IAM 用户指南》中的对于需要长期凭证的使用场景定期轮换访问密钥。

IAM 组是一个用于指定一组 IAM 用户的身份。您不能使用群组的身份登录。您可以使用组来一次性为多个用户指定权限。如果有大量用户，使用组可以更轻松地管理用户权限。例如，您可能具有一个名为 IAMAdmins 的组，并为该组授予权限以管理 IAM 资源。

用户与角色不同。用户唯一地与某个人员或应用程序关联，而角色旨在让需要它的任何人担任。用户具有永久的长期凭证，而角色提供临时凭证。要了解更多信息，请参阅《IAM 用户指南》中的何时创建 IAM 用户（而不是角色）。

IAM 角色

I AM 角色是您内部具有特定权限 Amazon Web Services 账户 的身份。它类似于 IAM 用户，但与特定人员不关联。您可以 Amazon Web Services Management Console 通过切换角色在中临时担任 IAM 角色。您可以通过调用 Amazon CLI 或 Amazon API 操作或使用自定义 URL 来代入角色。有关使用角色的方法的更多信息，请参阅《IAM 用户指南》中的使用 IAM 角色。

具有临时凭证的 IAM 角色在以下情况下很有用：

- 联合用户访问——要向联合身份分配权限，请创建角色并为角色定义权限。当联合身份进行身份验证时，该身份将与角色相关联并被授予由此角色定义的权限。有关联合身份验证的角色的信息，请参阅《IAM 用户指南》中的为第三方身份提供商创建角色。
- 临时 IAM 用户权限——IAM 用户或角色可代入 IAM 角色，以暂时获得针对特定任务的不同权限。
- 跨账户访问——您可以使用 IAM 角色以允许不同账户中的某个人（可信主体）访问账户中的资源。角色是授予跨账户存取权限的主要方式。但是，对于某些资源 Amazon Web Services，您可以将策略直接附加到资源（而不是使用角色作为代理）。要了解用于跨账户访问的角色和基于资源的策略之间的差别，请参阅《IAM 用户指南》中的 IAM 角色与基于资源的策略有何不同。
- 跨服务访问 — 有些 Amazon Web Services 使用其他 Amazon Web Services服务中的功能。例如，当您在某个服务中进行调用时，该服务通常会在 Amazon EC2 中运行应用程序或在 Amazon S3 中存储对象。服务可能会使用发出调用的主体的权限、使用服务角色或使用服务相关角色来执行此操作。
  - 转发访问会话 (FAS) — 当您使用 IAM 用户或角色在中执行操作时 Amazon，您被视为委托人。使用某些服务时，您可能会执行一个操作，此操作然后在不同服务中启动另一个操作。FAS 使用调

用委托人的权限以及 Amazon Web Service 向下游服务发出请求的请求。 Amazon Web Service 只有当服务收到需要与其他 Amazon Web Services 或资源交互才能完成的请求时，才会发出 FAS 请求。在这种情况下，您必须具有执行这两个操作的权限。有关发出 FAS 请求时的策略详情，请参阅转发访问会话。

- 服务角色 - 服务角色是服务代表您在您的账户中执行操作而分派的 IAM 角色。IAM 管理员可以在 IAM 中创建、修改和删除服务角色。有关更多信息，请参阅《IAM 用户指南》中的创建向 Amazon Web Service委派权限的角色。
- 服务相关角色-服务相关角色是一种链接到的服务角色。 Amazon Web Service服务可以担任代表您执行操作的角色。服务相关角色出现在您的中 Amazon Web Services 账户 ，并且归服务所有。IAM 管理员可以查看但不能编辑服务相关角色的权限。

- 在 A@@ mazon EC2 上运行的应用程序 — 您可以使用 IAM 角色管理在 EC2 实例上运行并发出 Amazon CLI 或 Amazon API 请求的应用程序的临时证书。这优先于在 EC2 实例中存储访问密钥。要向 EC2 实例分配 Amazon 角色并使其可供其所有应用程序使用，您需要创建附加到该实例的实例配置文件。实例配置文件包含角色，并使 EC2 实例上运行的程序能够获得临时凭证。有关更多信息，请参阅《IAM 用户指南》中的使用 IAM 角色为 Amazon EC2 实例上运行的应用程序授予权限。

要了解是使用 IAM 角色还是 IAM 用户，请参阅《IAM 用户指南》中的何时创建 IAM 角色（而不是用户）。

使用策略管理访问

您可以 Amazon 通过创建策略并将其附加到 Amazon 身份或资源来控制中的访问权限。策略是其中的一个对象 Amazon ，当与身份或资源关联时，它会定义其权限。 Amazon 在委托人（用户、root 用户或角色会话）发出请求时评估这些策略。策略中的权限确定是允许还是拒绝请求。大多数策略都以 JSON 文档的 Amazon 形式存储在中。有关 JSON 策略文档的结构和内容的更多信息，请参阅《IAM 用户指南》中的 JSON 策略概述。

管理员可以使用 Amazon JSON 策略来指定谁有权访问什么。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

默认情况下，用户和角色没有权限。要授予用户对所需资源执行操作的权限，IAM 管理员可以创建 IAM policy。然后，管理员可以向角色添加 IAM policy，并且用户可以代入角色。

IAM policy 定义操作的权限，无关乎您使用哪种方法执行操作。例如，假设有一个允许 `iam:GetRole` 操作的策略。拥有该策略的用户可以从 Amazon Web Services Management Console Amazon CLI、或 Amazon API 获取角色信息。

基于身份的策略

基于身份的策略是可附加到身份（如 IAM 用户、用户群组或角色）的 JSON 权限策略文档。这些策略控制用户和角色可在何种条件下对哪些资源执行哪些操作。要了解如何创建基于身份的策略，请参阅《IAM 用户指南》中的 创建 IAM policy。

基于身份的策略可以进一步归类为内联策略或托管式策略。内联策略直接嵌入单个用户、群组或角色中。托管策略是独立的策略，您可以将其附加到中的多个用户、群组和角色 Amazon Web Services 账户。托管策略包括 Amazon 托管策略和客户托管策略。要了解如何在托管式策略和内联策略之间进行选择，请参阅《IAM 用户指南》中的在托管式策略与内联策略之间进行选择。

基于资源的策略

基于资源的策略是附加到资源的 JSON 策略文档。基于资源的策略的示例包括 IAM 角色信任策略和 Amazon S3 存储桶策略。在支持基于资源的策略的服务中，服务管理员可以使用它们来控制对特定资源的访问。对于在其中附加策略的资源，策略定义指定主体可以对该资源执行哪些操作以及在什么条件下执行。您必须在基于资源的策略中指定主体。委托人可以包括账户、用户、角色、联合用户或 Amazon Web Services。

基于资源的策略是位于该服务中的内联策略。您不能在基于资源的策略中使用 IAM 中的 Amazon 托管策略。

访问控制列表 (ACL)

访问控制列表(ACL) 控制哪些主体（账户成员、用户或角色）有权访问资源。ACL 与基于资源的策略类似，尽管它们不使用 JSON 策略文档格式。

Amazon S3 和 Amazon VPC 就是支持 ACL 的服务示例。 Amazon WAF要了解有关 ACL 的更多信息，请参阅《Amazon Simple Storage Service 开发人员指南》中的访问控制列表 (ACL) 概述。

其他策略类型

Amazon 支持其他不太常见的策略类型。这些策略类型可以设置更常用的策略类型授予的最大权限。

- 权限边界——权限边界是一个高级特征，用于设置基于身份的策略可以为 IAM 实体（IAM 用户或角色）授予的最大权限。您可以为实体设置权限边界。这些结果权限是实体基于身份的策略及其权限边界的交集。在 `Principal` 中指定用户或角色的基于资源的策略不受权限边界限制。任一项策略中的显式拒绝将覆盖允许。有关权限边界的更多信息，请参阅《IAM 用户指南》中的 IAM 实体的权限边界。

- 服务控制策略 (SCP)-SCP 是 JSON 策略，用于指定组织或组织单位 (OU) 的最大权限。 Amazon Organizations Amazon Organizations 是一项用于对您的企业拥有的多 Amazon Web Services 账户项进行分组和集中管理的服务。如果您在组织内启用了特征，则可对任意或全部账户应用服务控制策略 (SCP)。SCP 限制成员账户中的实体（包括每个 Amazon Web Services 账户根用户实体）的权限。有关组织和 SCP 的更多信息，请参阅《Amazon Organizations 用户指南》中的 SCP 的工作原理。
- 会话策略——会话策略是当以编程方式为角色或联合用户创建临时会话时作为参数传递的高级策略。结果会话的权限是用户或角色的基于身份的策略和会话策略的交集。权限也可以来自基于资源的策略。任一项策略中的显式拒绝将覆盖允许。有关更多信息，请参阅《IAM 用户指南》中的会话策略。

多个策略类型

当多个类型的策略应用于一个请求时，生成的权限更加复杂和难以理解。要了解在涉及多种策略类型时如何 Amazon 确定是否允许请求，请参阅 IAM 用户指南中的策略评估逻辑。

如何 Amazon Web Services 使用 IAM

要全面了解如何 Amazon Web Services 使用大多数 IAM 功能，请参阅 IAM 用户指南中的与 IAM 配合使用的Amazon 服务。

要了解如何在 IAM 中 Amazon Web Service 使用特定的，请参阅相关服务的《用户指南》的安全部分。

对 Amazon 身份和访问进行故障排除

使用以下信息来帮助您诊断和修复在使用 Amazon 和 IAM 时可能遇到的常见问题。

主题

- 我无权在以下位置执行操作 Amazon
- 我无权执行 iam：PassRole
- 我想允许我以外的人 Amazon Web Services 账户 访问我的 Amazon 资源

我无权在以下位置执行操作 Amazon

如果您收到错误提示，表明您无权执行某个操作，则您必须更新策略以允许执行该操作。

当 `mateojackson` IAM 用户尝试使用控制台查看有关虚构 *`my-example-widget`* 资源的详细信息，但不拥有虚构 `awes:`*`GetWidget`* 权限时，会发生以下示例错误。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform: 
 awes:GetWidget on resource: my-example-widget
```

在此情况下，必须更新 `mateojackson` 用户的策略，以允许使用 `awes:`*`GetWidget`* 操作访问 *`my-example-widget`* 资源。

如果您需要帮助，请联系您的 Amazon 管理员。您的管理员是提供登录凭证的人。

我无权执行 iam：PassRole

如果您收到一个错误，表明您无权执行 `iam:PassRole` 操作，则必须更新策略以允许您将角色传递给 Amazon。

有些 Amazon Web Services 允许您将现有角色传递给该服务，而不是创建新的服务角色或服务相关角色。为此，您必须具有将角色传递到服务的权限。

当名为 `marymajor` 的 IAM 用户尝试使用控制台在 Amazon中执行操作时，会发生以下示例错误。但是，服务必须具有服务角色所授予的权限才可执行此操作。Mary 不具有将角色传递到服务的权限。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform: 
 iam:PassRole
```

在这种情况下，必须更新 Mary 的策略以允许她执行 `iam:PassRole` 操作。

如果您需要帮助，请联系您的 Amazon 管理员。您的管理员是提供登录凭证的人。

我想允许我以外的人 Amazon Web Services 账户 访问我的 Amazon 资源

您可以创建一个角色，以便其他账户中的用户或您组织外的人员可以使用该角色来访问您的资源。您可以指定谁值得信赖，可以代入角色。对于支持基于资源的策略或访问控制列表（ACL）的服务，您可以使用这些策略向人员授予对您的资源的访问权。

要了解更多信息，请参阅以下内容：

- 要了解是否 Amazon 支持这些功能，请参阅如何 Amazon Web Services 使用 IAM。
- 要了解如何提供对您拥有的资源的访问权限 Amazon Web Services 账户 ，请参阅 IAM 用户指南中的向您拥有 Amazon Web Services 账户 的另一个 IAM 用户提供访问权限。

- 要了解如何向第三方提供对您的资源的访问权限 Amazon Web Services 账户，请参阅 IAM 用户指南中的向第三方提供访问权限。 Amazon Web Services 账户
- 要了解如何通过身份联合验证提供访问权限，请参阅《IAM 用户指南》中的为经过外部身份验证的用户（身份联合验证）提供访问权限。
- 要了解使用角色和基于资源的策略进行跨账户存取之间的差别，请参阅《IAM 用户指南》中的 IAM 角色与基于资源的策略有何不同。

此 Amazon 产品或服务的合规性验证

要了解是否属于特定合规计划的范围，请参阅Amazon Web Services “” Amazon Web Services 中的 “按合规计划划分的范围”，然后选择您感兴趣的合规计划。 Amazon Web Service 有关一般信息，请参阅合规计划。

您可以使用下载第三方审计报告 Amazon Artifact。有关更多信息，请参阅中的 “下载报告” Amazon Artifact。

您在使用 Amazon Web Services 时的合规责任取决于您的数据的敏感性、贵公司的合规目标以及适用的法律和法规。 Amazon 提供了以下资源来帮助实现合规性：

- 安全与合规性快速入门指南 — 这些部署指南讨论了架构注意事项，并提供了在这些基础上 Amazon 部署以安全性和合规性为重点的基准环境的步骤。
- 合规资源 — 此工作簿和指南集合可能适用于您的行业和所在地区。
- 使用Amazon Config 开发人员指南中的规则评估资源 — 该 Amazon Config 服务评估您的资源配置在多大程度上符合内部实践、行业准则和法规。
- Amazon Security Hub— 这 Amazon Web Service 可以全面了解您的安全状态 Amazon。Security Hub 通过安全控件评估您的 Amazon 资源并检查其是否符合安全行业标准和最佳实践。有关受支持服务及控制的列表，请参阅 Security Hub 控制参考。

本 Amazon 产品或服务通过其支持的特定 Amazon Web Services (Amazon) 服务遵循分担责任模式。有关 Amazon 服务安全信息，请参阅Amazon 服务安全文档页面和合规计划合 Amazon 规工作范围内的Amazon 服务。

本 Amazon 产品或服务的弹性

Amazon 全球基础设施是围绕 Amazon Web Services 区域 可用区构建的。

Amazon Web Services 区域 提供多个物理隔离和隔离的可用区，这些可用区通过低延迟、高吞吐量和高度冗余的网络连接。

利用可用区，您可以设计和操作在可用区之间无中断地自动实现失效转移的应用程序和数据库。与传统的单个或多个数据中心基础设施相比，可用区具有更高的可用性、容错性和可扩展性。

有关 Amazon 区域和可用区的更多信息，请参阅Amazon 全球基础设施。

本 Amazon 产品或服务通过其支持的特定 Amazon Web Services (Amazon) 服务遵循分担责任模式。有关 Amazon 服务安全信息，请参阅Amazon 服务安全文档页面和合规计划合 Amazon 规工作范围内的Amazon 服务。

本 Amazon 产品或服务的基础设施安全

本 Amazon 产品或服务使用托管服务，因此受到 Amazon 全球网络安全的保护。有关 Amazon 安全服务以及如何 Amazon 保护基础设施的信息，请参阅Amazon 云安全。要使用基础设施安全的最佳实践来设计您的 Amazon 环境，请参阅 S Amazon ecurity Pillar Well-Architected Fram ework 中的基础设施保护。

您可以使用 Amazon 已发布的 API 调用通过网络访问此 Amazon 产品或服务。客户端必须支持以下内容：

- 传输层安全性协议 (TLS) 我们要求使用 TLS 1.2，建议使用 TLS 1.3。
- 具有完全向前保密 (PFS) 的密码套件，例如 DHE（临时 Diffie-Hellman）或 ECDHE（临时椭圆曲线 Diffie-Hellman）。大多数现代系统（如 Java 7 及更高版本）都支持这些模式。

此外，必须使用访问密钥 ID 和与 IAM 主体关联的秘密访问密钥来对请求进行签名。或者，您可以使用 Amazon Security Token Service（Amazon STS）生成临时安全凭证来对请求进行签名。

本 Amazon 产品或服务通过其支持的特定 Amazon Web Services (Amazon) 服务遵循分担责任模式。有关 Amazon 服务安全信息，请参阅Amazon 服务安全文档页面和合规计划合 Amazon 规工作范围内的Amazon 服务。

强制使用最低版本的 TLS

要在与 Amazon 服务通信时提高安全性，请将配置 Amazon SDK for JavaScript 为使用 TLS 1.2 或更高版本。

Important

Amazon SDK for JavaScript v3 会自动协商给定 Amazon 服务端点支持的最高级别 TLS 版本。您可以选择强制执行应用程序要求的最低 TLS 版本，例如 TLS 1.2 或 1.3，但请注意，某些 Amazon 服务端点不支持 TLS 1.3，因此，如果您强制执行 TLS 1.3，则某些调用可能会失败。

传输层安全性协议（TLS）是 Web 浏览器和其它应用程序使用的一种协议，用于确保通过网络交换的数据的隐私和完整性。

在 Node.js 中验证并强制执行 TLS

当你将 Node.js Amazon SDK for JavaScript 与一起使用时，底层 Node.js 安全层用于设置 TLS 版本。

Node.js 12.0.0 及更高版本使用支持 TLS 1.3 的最低版本 OpenSSL 1.1.1b。 Amazon SDK for JavaScript v3 在可用时默认使用 TLS 1.3，但如果需要，则默认为较低的版本。

验证 OpenSSL 和 TLS 的版本

要获取计算机上的 Node.js 使用的 OpenSSL 版本，请运行以下命令。

```
node -p process.versions
```

列表中的 OpenSSL 版本是 Node.js 使用的版本，如以下示例所示。

```
openssl: '1.1.1b'
```

要获取计算机上的 Node.js 使用的 TLS 版本，请启动 Node shell 并按顺序运行以下命令。

```
> var tls = require("tls");
> var tlsSocket = new tls.TLSSocket();
> tlsSocket.getProtocol();
```

最后一条命令输出 TLS 版本，如以下示例所示。

```
'TLSv1.3'
```

Node.js 默认使用此版本的 TLS，如果调用不成功，则会尝试协商另一个版本的 TLS。

强制使用最低版本的 TLS

当调用失败时，Node.js 会协商 TLS 的版本。您可以在此协商期间强制执行允许的最低 TLS 版本，无论是在命令行运行脚本时，还是在根据 JavaScript 代码中的请求运行脚本时。

要通过命令行指定最低 TLS 版本，必须使用 Node.js 版本 11.0.0 或更高版本。要安装特定的 Node.js 版本，请先按照 Node Version Manager Installing and Updating 中的步骤安装 Node Version Manager (nvm)。然后运行以下命令来安装并使用特定版本的 Node.js。

```
nvm install 11
nvm use 11
```

Enforce TLS 1.2

要强制规定 TLS 1.2 是允许的最低版本，请在运行脚本时指定 `--tls-min-v1.2` 参数，如以下示例所示。

```
node --tls-min-v1.2 yourScript.js
```

要在 JavaScript 代码中为特定请求指定允许的最低 TLS 版本，请使用`httpOptions`参数指定协议，如以下示例所示。

```
import https from "https";
import { NodeHttpHandler } from "@smithy/node-http-handler";
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({
    region: "us-west-2",
    requestHandler: new NodeHttpHandler({
        httpsAgent: new https.Agent(
            {
                secureProtocol: 'TLSv1_2_method'
            }
        )
    })
});
```

Enforce TLS 1.3

要强制规定 TLS 1.3 是允许的最低版本，请在运行脚本时指定 `--tls-min-v1.3` 参数，如以下示例所示。

```
node --tls-min-v1.3 yourScript.js
```

要在 JavaScript 代码中为特定请求指定允许的最低 TLS 版本，请使用`httpOptions`参数指定协议，如以下示例所示。

```
import https from "https";
import { NodeHttpHandler } from "@smithy/node-http-handler";
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({
    region: "us-west-2",
    requestHandler: new NodeHttpHandler({
        httpsAgent: new https.Agent(
            {
                secureProtocol: 'TLSv1_3_method'
            }
        )
    })
});
```

在浏览器脚本中验证并强制执行 TLS

当您在浏览器脚本 JavaScript 中使用 SDK 时，浏览器设置会控制所使用的 TLS 版本。浏览器使用的 TLS 版本无法通过脚本发现或设置，必须由用户配置。要验证和强制执行浏览器脚本中使用的 TLS 版本，请参阅特定浏览器的说明。

Microsoft Internet Explorer

1. 打开 Internet Explorer。
2. 从菜单栏中选择工具 - Internet 选项 - 高级选项卡。
3. 向下滚动到安全类别，手动选中使用 TLS 1.2 选项框。
4. 单击确定。
5. 关闭浏览器并重新启动 Internet Explorer。

Microsoft Edge

1. 在 Windows 菜单搜索框中，键入 *`Internet options`*。
2. 在最佳匹配下，单击 Internet 选项。

3. 在 Internet 属性窗口的高级选项卡上，向下滚动到安全部分。
4. 选中用户 TLS 1.2 复选框。
5. 单击 确定。

Google Chrome

1. 打开 Google Chrome。
2. 按 Alt F 并选择设置。
3. 向下滚动并选择显示高级设置...。
4. 向下滚动到系统部分，然后单击打开代理设置...。
5. 选择高级选项卡。
6. 向下滚动到安全类别，手动选中使用 TLS 1.2 选项框。
7. 单击确定。
8. 关闭浏览器并重启 Google Chrome。

Mozilla Firefox

1. 打开 Firefox。
2. 在地址栏中键入 about:config，然后按 Enter。
3. 在搜索字段中输入 tls。找到并双击 security.tls.version.min 条目。
4. 将整数值设置为 3 以强制将 TLS 1.2 协议设为默认协议。
5. 单击 确定。
6. 关闭浏览器并重启 Mozilla Firefox。

Apple Safari

没有启用 SSL 协议的选项。如果您使用的是 Safari 浏览器 7 或更高版本，则会自动启用 TLS 1.2。

迁移到版本 3

本节介绍如何从的版本 2 迁移到版本 3 Amazon SDK for JavaScript。

将您的代码迁移到 JavaScript V3 版 SDK

Amazon SDK for JavaScript 版本 3 (v3) 带有用于客户端配置和实用程序的现代化界面，包括凭证、Amazon S3 分段上传、DynamoDB 文档客户端、服务员等。您可以在 repo 的迁移指南中找到 v2 中更改的内容以及每项更改的 v3 等效内容。 Amazon SDK for JavaScript GitHub

要充分利用 Amazon SDK for JavaScript v3，我们建议使用下面描述的 codemod 脚本。

使用 codemod 迁移现有的 v2 代码

中的 codemod 脚本集aws-sdk-js-codemod有助于将现有的 Amazon SDK for JavaScript (v2) 应用程序迁移到使用 v3 API。您可以按以下方式运行转换。

```
$ npx aws-sdk-js-codemod -t v2-to-v3 PATH...
```

例如，假设您有以下代码，它用于从 v2 创建一个 Amazon DynamoDB 客户端并调用 `listTables` 运算。

```
// example.ts
import AWS from "aws-sdk";

const region = "us-west-2";
const client = new AWS.DynamoDB({ region });
client.listTables({}, (err, data) => {
  if (err) console.log(err, err.stack);
  else console.log(data);
});
```

你可以按如下方式对 `example.ts` 运行我们的 `v2-to-v3` 转换。

```
$ npx aws-sdk-js-codemod -t v2-to-v3 example.ts
```

转换会将 DynamoDB 导入转换为 v3，创建 v3 客户端，然后调用 `listTables` 运算，如下所示。

```
// example.ts
```

```
import { DynamoDB } from "@aws-sdk/client-dynamodb";

const region = "us-west-2";
const client = new DynamoDB({ region });
client.listTables({}, (err, data) => {
  if (err) console.log(err, err.stack);
  else console.log(data);
});
```

我们已经针对常见使用案例实现了转换。如果您的代码无法正确转换，请使用示例输入代码和观察到/预期的输出代码创建错误报告或功能请求。如果某个现有问题已经报告了您的特定使用案例，请通过投赞同票表示支持。

Amazon SDK for JavaScript 版本 3 的文档历史记录

文档历史记录

下表介绍了 2020 年 10 月 20 日之后 Amazon SDK for JavaScript V3 版本中的重要更改。如需获取对此文档的更新的通知，您可以订阅 RSS 源。

变更	说明	日期
公告	更新了顶部横幅，其中包含了 Internet Explorer 11 end-of-support 提醒	2022 年 9 月 23 日
次要更新	对明确性和修复损坏的链接进行了小幅更新。添加了指向 Amazon SDK 和 “工具参考指南” 的感知链接。	2022 年 8 月 22 日
强制使用最低 TLS 版本	添加了有关 TLS 1.3 的信息。	2022 年 3 月 31 日
更新了 Amazon Lambda 教程	添加了演示如何构建用于将数据提交到 Amazon DynamoDB 表的基于浏览器的应用程序的教程。	2020 年 10 月 20 日
在 Node.js 中设置凭证主题已更新	更新有关在 Node.js 中为 Amazon SDK for JavaScript V3 设置凭据的主题。	2020 年 10 月 20 日
迁移到 V3	添加了描述如何迁移到 Amazon SDK for JavaScript V3 的主题。	2020 年 10 月 20 日
开始使用	更新了浏览器入门和开始使用适用于 Amazon SDK for JavaScript V3 的 Node.js 的主题。	2020 年 10 月 20 日

浏览器生成器	Amazon 浏览器生成器的相关信息已被删除，因为 Amazon SDK for JavaScript V3 不需要这些信息。	2020 年 10 月 20 日
更新了 Amazon Transcribe 服务示例	更新了 V3 的 Amazon Transcribe 服务示例。Amazon SDK for JavaScript	2020 年 10 月 20 日
更新了 Amazon Simple Notification Service 服务示例	更新了 Amazon SDK for JavaScript V3 的 Amazon 简单通知服务示例。	2020 年 10 月 20 日
更新了 Amazon Simple Email Service 服务示例	更新了 Amazon SDK for JavaScript V3 的 Amazon 简单电子邮件服务示例。	2020 年 10 月 20 日
更新了 Amazon Redshift 服务示例	更新了 V3 的亚马逊 Redshift 服务示例。 Amazon SDK for JavaScript	2020 年 10 月 20 日
更新了 Amazon Lex 服务示例	更新了 Amazon SDK for JavaScript V3 的 Amazon Lex 服务示例。	2020 年 10 月 20 日
更新了 Amazon DynamoDB 服务示例	更新了 V3 的亚马逊 DynamoDB 服务示例。Amazon SDK for JavaScript	2020 年 10 月 20 日
AWS Elemental MediaConvert 服务示例已更新	更新了 Amazon SDK for JavaScript V3 的 AWS Elemental MediaConvert 服务示例。	2020 年 10 月 20 日
Amazon Lambda 服务示例已更新	更新了 Amazon SDK for JavaScript V3 的 Amazon Lambda 服务示例。	2020 年 10 月 20 日

Amazon SDK for JavaScript V3 开发者指南预览	已发布 Amazon SDK for JavaScript V3 开发者指南的预发行版。	2020 年 10 月 19 日