亚马逊云科技

开发人员指南

Amazon Serverless Application Model

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon Serverless Application Model: 开发人员指南

Copyright © 2025 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

什么是 Amazon SAM？ ... 1
- 主要特征 ... 1
- 相关信息 ... 2
- 工作方式 ... 2
  - 什么是 Amazon SAM 模板规范？ ... 2
  - Amazon SAM 项目和 Amazon SAM 模板是什么？ ... 3
  - 那是什么 Amazon SAMCLI？ ... 9
  - 了解更多 ... 17
  - 后续步骤 ... 17
- 无服务器概念 ... 17
  - 无服务器概念 ... 18

入门 ... 19
- 先决条件 ... 19
  - 第 1 步：注册一个 Amazon 账户 ... 20
  - 第 2 步：创建 IAM 用户账户 ... 20
  - 第 3 步：创建访问密钥 ID 和秘密访问密钥 ... 20
  - 步骤 4：安装 Amazon CLI ... 21
  - 步骤 5： Amazon CLI 使用配置 Amazon 凭证 ... 21
  - 后续步骤 ... 22
- 安装 Amazon SAM CLI ... 22
  - 安装 Amazon SAM CLI ... 23
  - 排查 安装错误 ... 32
  - 后续步骤 ... 34
  - 可选：验证 Amazon SAMCLI 安装程序 ... 34
- Hello World 教程 ... 46
  - 先决条件 ... 47
  - 第 1 步：初始化示例 Hello World 应用程序 ... 47
  - 第 2 步：构建应用程序 ... 51
  - 步骤 3：将您的应用程序部署到 Amazon Web Services 云 ... 53
  - 第 4 步：运行应用程序 ... 57
  - 第 5 步：在中与你的函数进行交互 Amazon Web Services 云 ... 58
  - 步骤 6：修改您的应用程序并将其同步到 Amazon Web Services 云 ... 59
  - 第 7 步：（可选）在本地测试应用程序 ... 63
  - 第 8 步：从中删除您的应用程序 Amazon Web Services 云 ... 65

故障排除 65
了解更多 65
如何使用 Amazon SAM 67
的 Amazon SAM CLI 67
怎么样 Amazon SAM CLI 命令已记录在案 68
正在配置 Amazon SAM CLI 69
核心命令 74
该 Amazon SAM 项目 76
模板剖析 77
资源和属性 85
生成的资源 407
支持的资源属性 423
API Gateway 扩展 425
内置函数 426
开发应用程序 427
创建应用程序 427
初始化新的无服务器应用程序 428
sam init 的选项 433
故障排除 433
示例 434
了解更多 434
后续步骤 434
定义基础设施 435
定义应用程序资源 435
设置访问 436
控制 API 访问 516
使用层提高效率 528
重复使用代码 531
管理基于时间的事件 534
编排应用程序 537
设置代码签名 538
验证 Amazon SAM 模板文件 541
构建应用程序 541
简介 sam build 542
默认构建 555
自定义构建 562

测试您的应用程序 586
简介 sam local 586
使用 sam local 命令 587
简介 sam local generate-event 587
简介 sam local invoke 593
简介 sam local start-api 599
简介 sam local start-lambda 604
本地调用函数 607
环境变量文件 607
图层 609
了解更多 609
在本地运行 API Gateway 609
环境变量文件 610
图层 612
使用以下方法进行测试 sam remote test-event 612
设置 Amazon SAM CLI 要使用 sam remote test-event 613
使用 sam remote test-event 命令 613
使用可共享测试事件 616
管理可共享测试事件 616
使用以下方法进行测试 sam remote invoke 617
使用 sam remote invoke 命令 618
使用 sam remote invoke 命令选项 622
配置您的项目配置文件 627
示例 627
相关链接 642
自动集成测试 643
生成示例有效载荷 644
调试应用程序 646
在本地调试函数 646
使用 Amazon 工具包 647
在调试模式下在 Amazon SAM 本地运行 648
传递多个运行时参数 649
使用 cfn-lint 进行验证 649
示例 650
了解更多 650
部署应用程序和资源 651

简介 sam deploy 651
先决条件 652
使用 sam deploy 部署应用程序 652
最佳实践 662
sam deploy 的选项 662
故障排除 662
示例 662
了解更多 671
部署选项 671
如何使用 Amazon SAM CLI 手动部署 671
使用 CI/CD 系统和管线进行部署 672
逐步部署 672
使用对部署进行故障排除 Amazon SAM CLI 672
了解更多 609
使用 CI/CD 系统和管线进行部署 673
什么是管线？ 674
如何 Amazon SAM 上传本地文件 674
生成入门管线 682
自定义入门管线 687
自动部署 689
使用 OIDC 身份验证 693
简介 sam sync 695
自动检测本地更改并将其同步到 Amazon Web Services 云 696
自定义将哪些本地更改同步到 Amazon Web Services 云 697
为云端应用程序做好测试和验证准备 698
sam sync 命令的选项 698
故障排除 700
示例 700
了解更多 707
监控应用程序 708
Application Insights 708
使用配置 CloudWatch 应用程序见解 Amazon SAM 708
后续步骤 712
使用日志 712
按堆栈获取日 Amazon CloudFormation 志 712
通过 Lambda 函数名称获取日志 712

跟踪日志 712
查看特定时间范围的日志 713
筛选日志 713
错误突出显示 713
JSON 整齐打印 713
Amazon SAM 参考 714
Amazon SAM 规格和 Amazon SAM 模板 714
Amazon SAM CLI 命令参考 714
Amazon SAM 策略模板 715
主题 715
Amazon SAM CLI 命令 715
sam build 716
sam delete 721
sam deploy 722
sam init 728
sam list 731
sam local generate-event 738
sam local invoke 740
sam local start-api 745
sam local start-lambda 749
sam logs 754
sam package 757
sam pipeline bootstrap 760
sam pipeline init 764
sam publish 765
sam remote invoke 767
sam remote test-event 772
sam sync 778
sam traces 784
sam validate 786
Amazon SAM CLI management 787
Amazon SAM CLI 配置文件 787
管理 Amazon SAM CLI 版本 794
设置 Amazon 凭证 803
Amazon SAM CLI 遥测功能 804
故障排除 806

连接器参考 811
支持的连接器资源类型 812
连接器创建的 IAM 策略 821
安装 Docker 844
安装 Docker 845
后续步骤 848
映像存储库 848
镜像存储库 URIs 848
示例 850
逐步部署 850
首次逐步部署 Lambda 函数 853
了解更多 854
重要提示 854
2023 854
示例应用程序 856
处理 DynamoDB 事件 856
开始前的准备工作 856
第 1 步：初始化应用程序 856
第 2 步：在本地测试应用程序 857
第 3 步：打包应用程序 857
第 4 步：部署应用程序 858
后续步骤 858
处理 Amazon S3 事件 858
开始前的准备工作 859
第 1 步：初始化应用程序 859
第 2 步：打包应用程序 860
第 3 步：部署应用程序 860
第 4 步：在本地测试应用程序 861
后续步骤 861
Terraform 支持 862
入门 862
先决条件 862
将 Amazon SAM CLI 命令与 Terraform 结合使用 863
为 Terraform 项目做好准备 863
设置 Terraform Cloud 868
结合使用 Amazon SAM CLI 和 Terraform 870

使用 sam local invoke 进行本地测试 870
使用 sam local start-api 进行本地测试 870
使用 sam local start-lambda 进行本地测试 872
Terraform 限制 873
将 Amazon SAM CLI 与 Serverless.tf 一起使用 873
Terraform 参考 873
Amazon SAM 支持的功能参考 874
Terraform 特定参考 874
sam 元数据 874
Amazon SAM CLI Terraform 支持 877
那是什么 Amazon SAMCLI？ 877
如何结合使用 Amazon SAM CLI 和 Terraform？ 878
后续步骤 878
发布供他人使用 879
先决条件 879
发布新应用程序 880
第 1 步：向 Amazon SAM 模板添加分Metadata区 880
第 2 步：打包应用程序 881
第 3 步：发布应用程序 882
第 4 步：共享应用程序（可选） 882
发布现有应用程序的新版本 882
其他主题 882
元数据部分属性 883
属性 883
使用案例 885
示例 886
文档历史记录 887
..... cmv

Amazon Serverless Application Model (Amazon SAM) 是什么？

Amazon Serverless Application Model （Amazon SAM）是一个使用基础设施即代码（IaC）构建无服务器应用程序的开源框架。使用 Amazon SAM的简写语法，开发人员可以声明Amazon CloudFormation资源和专业的无服务器资源，这些资源在部署期间会转换为基础设施。此框架包括两个主要组件： Amazon SAMCLI和 Amazon SAM 项目。 Amazon SAM 项目是在您运行时创建的应用程序项目目录sam init。 Amazon SAM 项目包含 Amazon SAM 模板之类的文件，其中包含模板规范（用于声明资源的简写语法）。

主要特征

Amazon SAM 提供多种好处，通过允许您执行以下操作来改善开发人员体验：

使用更少的代码快速定义应用程序基础设施代码

编 Amazon SAM 写模板来定义无服务器应用程序基础设施代码。将模板直接部署到 Amazon CloudFormation 以配置资源。

在无服务器应用程序的整个开发生命周期中对其进行管理

使用 Amazon SAM CLI 在开发生命周期的编写、构建、部署、测试和监控阶段管理无服务器应用程序。有关更多信息，请参阅 的 Amazon SAM CLI。

使用 Amazon SAM 连接器快速配置资源之间的权限

在 Amazon SAM 模板中使用 Amazon SAM 连接器来定义 Amazon 资源之间的权限。 Amazon SAM 会将代码转换为所需的 IAM 权限，以帮助实现您的目的。有关更多信息，请参阅 使用 Amazon SAM 连接器管理资源权限。

在开发时持续将本地更改同步到云端

使用 Amazon SAMCLIsam sync命令自动将本地更改同步到云端，从而加快开发和云测试工作流程。有关更多信息，请参阅 使用简介 sam sync 同步到 Amazon Web Services 云。

管理 Terraform 无服务器应用程序

使用 Amazon SAM CLI 对 Lambda 函数和层执行本地调试和测试。有关更多信息，请参阅 Amazon SAM CLI Terraform 支持。

相关信息

- 有关 Amazon SAM 工作原理的信息，请参阅如何 Amazon SAM 运作。
- 要开始使用 Amazon SAM，请参阅入门 Amazon SAM。
- 有关如何使用创建无服务器应用程序 Amazon SAM 的概述，请参阅如何使用 Amazon SAM。

如何 Amazon SAM 运作

Amazon SAM 由用于创建无服务器应用程序的两个主要组件组成：

1. 该 Amazon SAM 项目 – 运行 sam init 命令时创建的文件夹和文件。此目录包含Amazon SAM 模板，这是定义您的 Amazon 资源的重要文件。此模板包含Amazon SAM 模板规范 — 附带简化的简写语法的开源框架，您可以使用它来定义无服务器应用程序的函数 APIs、事件、配置和权限。
2. 的 Amazon SAM CLI— 一个命令行工具，可与 Amazon SAM 项目和受支持的第三方集成结合使用，用于构建和运行无服务器应用程序。 Amazon SAMCLI是用于在 Amazon SAM 项目上运行命令并最终将其转换为无服务器应用程序的工具。

要表达定义无服务器应用程序的资源、事件源映射和其他属性，您需要在 Amazon SAM 模板和项目中的其他文件中定义资源并开发应用程序。 Amazon SAM 您可以使用在 Amazon SAMCLI Amazon SAM 项目上运行命令，这就是初始化、构建、测试和部署无服务器应用程序的方式。

> 无服务器技术的新手？
>
> 我们建议您查看的无服务器概念 Amazon Serverless Application Model。

什么是 Amazon SAM 模板规范？

Amazon SAM 模板规范是一个开源框架，可用于定义和管理无服务器应用程序基础设施代码。Amazon SAM 模板规范具有以下特点：

- 基于构建 Amazon CloudFormation-您可以直接在 Amazon SAM 模板中使用 Amazon CloudFormation 语法，利用其对资源和属性配置的广泛支持。如果您已经熟悉 Amazon CloudFormation，则无需了解新服务即可管理应用程序基础设施代码。
- 是的扩展 Amazon CloudFormation- Amazon SAM 提供自己独有的语法，专门用于加快无服务器开发。可以在同一个模板中使用 Amazon CloudFormation 和 Amazon SAM 语法。

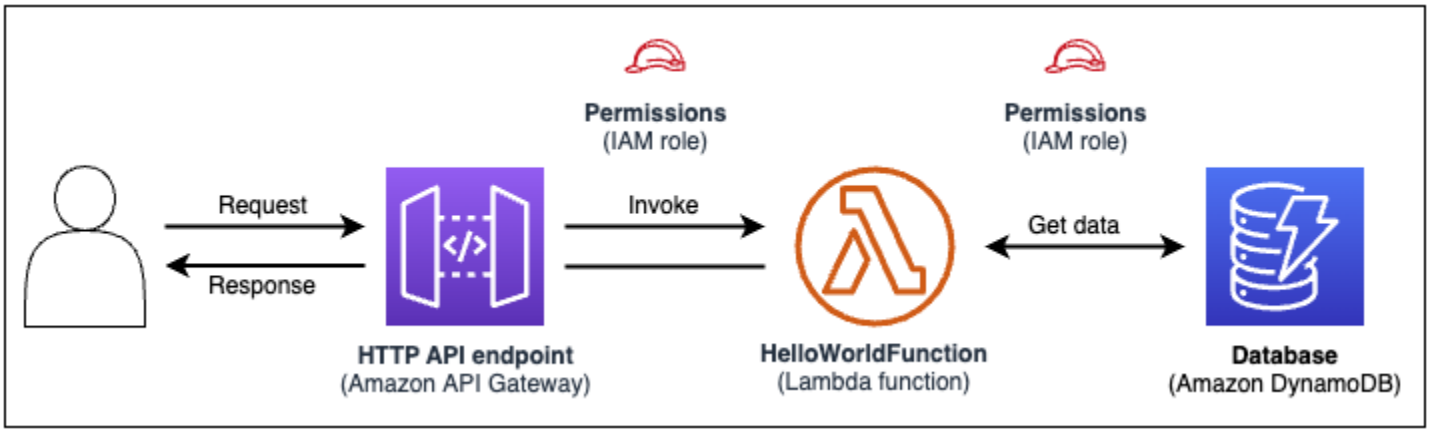
- 是一种抽象的速记语法 - 借助 Amazon SAM 语法，您可以用更少的代码行快速定义基础设施，而且出错的可能性更低。它的语法经过特别精心设计，可消除定义无服务器应用程序基础设施的复杂性。
- 可@@ 转换- Amazon SAM 通过 Amazon CloudFormation来完成将模板转换为配置基础设施所需的代码这项复杂工作。

Amazon SAM 项目和 Amazon SAM 模板是什么？

Amazon SAM 项目包括 Amazon SAM 模板，其中包含 Amazon SAM 模板规范。此规范是用于在上定义无服务器应用程序基础设施的开源框架 Amazon，并且附带一些使其更易于使用的其他组件。从这个意义上讲， Amazon SAM 模板是 Amazon CloudFormation 模板的扩展。

以下是基本无服务器应用程序的示例：此应用程序处理想要通过 HTTP 请求从数据库获取所有项目的请求。它由以下几个部分组成：

1. 包含用于处理请求的逻辑的函数。
2. HTTP API，用作客户端（请求程序）和应用程序之间的通信。
3. 用于存储项目的数据库。
4. 应用程序安全运行所需的权限。



可以在以下 Amazon SAM 模板中定义此应用程序的基础设施代码：

```
AWSTemplateFormatVersion: 2010-09-09
Transform: AWS::Serverless-2016-10-31
Resources:
  getAllItemsFunction:
    Type: AWS::Serverless::Function
    Properties:
```

```
      Handler: src/get-all-items.getAllItemsHandler
      Runtime: nodejs20.x
      Events:
        Api:
          Type: HttpApi
          Properties:
            Path: /
            Method: GET
    Connectors:
      MyConn:
        Properties:
          Destination:
            Id: SampleTable
          Permissions:
            - Read
  SampleTable:
    Type: AWS::Serverless::SimpleTable
```

在 23 行代码中，定义了以下基础设施：

- 使用该 Amazon Lambda 服务的函数。
- 使用 Amazon API Gateway 服务的 HTTP API。
- 使用 Amazon DynamoDB 服务的数据库。
- 这些服务互相交互所需的 Amazon Identity and Access Management (IAM) 权限。

要配置此基础设施，需要将模板部署到 Amazon CloudFormation。在部署过程中， Amazon SAM 会将 23 行代码转换为在中生成这些资源所需的 Amazon CloudFormation 语法。 Amazon转换后的 Amazon CloudFormation 模板包含 200 多行代码！

转换后的 Amazon CloudFormation 模板

```
{
  "AWSTemplateFormatVersion": "2010-09-09",
  "Resources": {
    "getAllItemsFunction": {
      "Type": "AWS::Lambda::Function",
      "Metadata": {
        "SamResourceId": "getAllItemsFunction"
      },
      "Properties": {
        "Code": {
```

```
          "S3Bucket": "amzn-s3-demo-source-bucket-1a4x26zbcdkqr",
          "S3Key": "what-is-app/a6f856abf1b2c4f7488c09b367540b5b"
        },
        "Handler": "src/get-all-items.getAllItemsHandler",
        "Role": {
          "Fn::GetAtt": [
            "getAllItemsFunctionRole",
            "Arn"
          ]
        },
        "Runtime": "nodejs12.x",
        "Tags": [
          {
            "Key": "lambda:createdBy",
            "Value": "SAM"
          }
        ]
      }
    },
    "getAllItemsFunctionRole": {
      "Type": "AWS::IAM::Role",
      "Properties": {
        "AssumeRolePolicyDocument": {
          "Version": "2012-10-17",
          "Statement": [
            {
              "Action": [
                "sts:AssumeRole"
              ],
              "Effect": "Allow",
              "Principal": {
                "Service": [
                  "lambda.amazonaws.com"
                ]
              }
            }
          ]
        },
        "ManagedPolicyArns": [
          "arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole"
        ],
        "Tags": [
          {
            "Key": "lambda:createdBy",
```

```
            "Value": "SAM"
          }
        ]
      }
    },
    "getAllItemsFunctionApiPermission": {
      "Type": "AWS::Lambda::Permission",
      "Properties": {
        "Action": "lambda:InvokeFunction",
        "FunctionName": {
          "Ref": "getAllItemsFunction"
        },
        "Principal": "apigateway.amazonaws.com",
        "SourceArn": {
          "Fn::Sub": [
            "arn:${AWS::Partition}:execute-api:${AWS::Region}:${AWS::AccountId}:
${__ApiId__}/${__Stage__}/GET/",
            {
              "__ApiId__": {
                "Ref": "ServerlessHttpApi"
              },
              "__Stage__": "*"
            }
          ]
        }
      }
    },
    "ServerlessHttpApi": {
      "Type": "AWS::ApiGatewayV2::Api",
      "Properties": {
        "Body": {
          "info": {
            "version": "1.0",
            "title": {
              "Ref": "AWS::StackName"
            }
          },
          "paths": {
            "/": {
              "get": {
                "x-amazon-apigateway-integration": {
                  "httpMethod": "POST",
                  "type": "aws_proxy",
                  "uri": {
```

```
                    "Fn::Sub": "arn:${AWS::Partition}:apigateway:
${AWS::Region}:lambda:path/2015-03-31/functions/${getAllItemsFunction.Arn}/invocations"
                  },
                  "payloadFormatVersion": "2.0"
                },
                "responses": {}
              }
            }
          },
          "openapi": "3.0.1",
          "tags": [
            {
              "name": "httpapi:createdBy",
              "x-amazon-apigateway-tag-value": "SAM"
            }
          ]
        }
      }
    },
    "ServerlessHttpApiApiGatewayDefaultStage": {
      "Type": "AWS::ApiGatewayV2::Stage",
      "Properties": {
        "ApiId": {
          "Ref": "ServerlessHttpApi"
        },
        "StageName": "$default",
        "Tags": {
          "httpapi:createdBy": "SAM"
        },
        "AutoDeploy": true
      }
    },
    "SampleTable": {
      "Type": "AWS::DynamoDB::Table",
      "Metadata": {
        "SamResourceId": "SampleTable"
      },
      "Properties": {
        "AttributeDefinitions": [
          {
            "AttributeName": "id",
            "AttributeType": "S"
          }
        ],
```

```
        "KeySchema": [
          {
            "AttributeName": "id",
            "KeyType": "HASH"
          }
        ],
        "BillingMode": "PAY_PER_REQUEST"
      }
    },
    "getAllItemsFunctionMyConnPolicy": {
      "Type": "AWS::IAM::ManagedPolicy",
      "Metadata": {
        "aws:sam:connectors": {
          "getAllItemsFunctionMyConn": {
            "Source": {
              "Type": "AWS::Serverless::Function"
            },
            "Destination": {
              "Type": "AWS::Serverless::SimpleTable"
            }
          }
        }
      },
      "Properties": {
        "PolicyDocument": {
          "Version": "2012-10-17",
          "Statement": [
            {
              "Effect": "Allow",
              "Action": [
                "dynamodb:GetItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:BatchGetItem",
                "dynamodb:ConditionCheckItem",
                "dynamodb:PartiQLSelect"
              ],
              "Resource": [
                {
                  "Fn::GetAtt": [
                    "SampleTable",
                    "Arn"
                  ]
                },
```

```
                {
                  "Fn::Sub": [
                    "${DestinationArn}/index/*",
                    {
                      "DestinationArn": {
                        "Fn::GetAtt": [
                          "SampleTable",
                          "Arn"
                        ]
                      }
                    }
                  ]
                }
              ]
            }
          ]
        },
        "Roles": [
          {
            "Ref": "getAllItemsFunctionRole"
          }
        ]
      }
    }
  }
}
```

通过使用 Amazon SAM，您可以定义 23 行基础架构代码。 Amazon SAM 会将代码转换为配置应用程序所需的 200 多行 Amazon CloudFormation 代码。

那是什么 Amazon SAMCLI？

Amazon SAMCLI是一个命令行工具，可与 Amazon SAM 模板和受支持的第三方集成结合使用，用于构建和运行无服务器应用程序。使用 Amazon SAM CLI 来：

- 快速初始化新的应用程序项目。
- 构建应用程序以进行部署。
- 执行本地调试和测试。
- 部署您的应用程序。
- 配置 CI/CD 部署管道。
- 对云端应用程序进行监控和问题排查。

- 在开发时将本地更改同步到云端。
- 还有更多！

与 Amazon SAM 和 Amazon CloudFormation 模板一起使用时最能利用。 Amazon SAMCLI它还可与 Terraform 等第三方产品配合使用。

初始化新项目

从入门模板中进行选择或选择自定义模板位置以开始新项目。

在这里，我们使用 sam init 命令来初始化新的应用程序项目。首先选择 Hello World 示例项目。Amazon SAM CLI 会下载入门模板并创建项目文件夹目录结构。

```
➜  what-is sam init

You can preselect a particular runtime or package type when using the `sam init` experience.
Call `sam init --help` to learn more.

Which template source would you like to use?
        1 - AWS Quick Start Templates
        2 - Custom Template Location
Choice: 1

Choose an AWS Quick Start application template
        1 - Hello World Example
        2 - Multi-step workflow
        3 - Serverless API
        4 - Scheduled task
        5 - Standalone function
        6 - Data processing
        7 - Infrastructure event management
        8 - Serverless Connector Hello World Example
        9 - Multi-step workflow with Connectors
        10 - Lambda EFS example
        11 - Machine Learning
Template: 1

Use the most popular runtime and package type? (Python and zip) [y/N]:
```

有关更多详细信息，请参阅 在中创建您的应用程序 Amazon SAM。

构建应用程序以进行部署

打包函数依赖项并整理项目代码和文件夹结构，为部署做准备。

在这里，我们使用 sam build 命令来准备应用程序以进行部署。 Amazon SAM CLI 会创建 `.aws-sam` 目录并在其中整理应用程序依赖项和文件，以进行部署。

```
➜  sam-app sam build
Building codeuri: /Users/evzz/Demo/what-is/sam-app/hello_world runtime: python3.9 metadata: {} archit
ecture: x86_64 functions: HelloWorldFunction
Running PythonPipBuilder:ResolveDependencies
Running PythonPipBuilder:CopySource

Build Succeeded

Built Artifacts  : .aws-sam/build
Built Template   : .aws-sam/build/template.yaml

Commands you can use next
=========================
[*] Validate SAM template: sam validate
[*] Invoke Function: sam local invoke
[*] Test Function in the Cloud: sam sync --stack-name {{stack-name}} --watch
[*] Deploy: sam deploy --guided
➜  sam-app cd .aws-sam
➜  .aws-sam ls
build      build.toml
➜  .aws-sam
```

有关更多详细信息，请参阅 构建应用程序。

执行本地调试和测试。

在本地计算机上模拟事件，测试 APIs，调用函数以及进行其他操作，以调试和测试应用程序。

在这里，我们使用 sam local invoke 命令在本地调用 `HelloWorldFunction`。为此， Amazon SAM CLI 会创建本地容器，构建函数，调用函数，然后输出结果。您可以使用 Docker 之类的应用程序在计算机上运行容器。

```
➜  sam-app sam local invoke HelloWorldFunction
Invoking app.lambda_handler (python3.9)
Local image was not found.
Removing rapid images for repo public.ecr.aws/sam/emulation-python3.9
Building image.....................................................................................
.....................................................................................................
.............................................................
Using local image: public.ecr.aws/lambda/python:3.9-rapid-x86_64.

Mounting /Users/evzz/Demo/what-is/sam-app/.aws-sam/build/HelloWorldFunction as /var/task:ro,delegated
 inside runtime container
START RequestId: 6f8347ce-6b04-4246-a0de-6dc37f0eef51 Version: $LATEST
END RequestId: 6f8347ce-6b04-4246-a0de-6dc37f0eef51
REPORT RequestId: 6f8347ce-6b04-4246-a0de-6dc37f0eef51  Init Duration: 1.23 ms  Duration: 639.26 ms B
illed Duration: 640 ms  Memory Size: 128 MB     Max Memory Used: 128 MB
{"statusCode": 200, "body": "{\"message\": \"hello world\"}"}
```

有关更多详细信息，请参阅 测试您的应用程序 和 调试应用程序。

部署您的应用程序

配置应用程序的部署设置并将应用程序部署到 Amazon Cloud，以配置资源。

在这里，我们使用 sam deploy --guided 命令通过交互式流程部署应用程序。会 Amazon SAMCLI指导我们配置应用程序的部署设置，将模板转换为 Amazon CloudFormation，并将应用程序部署到，Amazon CloudFormation 以创建资源。

```
➜  sam-app sam deploy --guided

Configuring SAM deploy
======================

        Looking for config file [samconfig.toml] :  Not found

        Setting default arguments for 'sam deploy'
        =========================================
        Stack Name [sam-app]:
        AWS Region [us-west-2]:
        #Shows you resources changes to be deployed and require a 'Y' to initiate deploy
        Confirm changes before deploy [y/N]:
        #SAM needs permission to be able to create roles to connect to the resources in your template
        Allow SAM CLI IAM role creation [Y/n]:
        #Preserves the state of previously provisioned resources when an operation fails
        Disable rollback [y/N]:
        HelloWorldFunction may not have authorization defined, Is this okay? [y/N]: y
        Save arguments to configuration file [Y/n]:
        SAM configuration file [samconfig.toml]:
        SAM configuration environment [default]:

        Looking for resources needed for deployment:
         Managed S3 bucket: aws-sam-cli-managed-default-samclisourcebucket-1a4x26zbcdkqr
         A different default S3 bucket can be set in samconfig.toml
```

有关更多详细信息，请参阅 部署应用程序和资源。

配置 CI/CD 部署管道

使用受支持的 CI/CD 系统创建安全的持续集成和持续交付 (CI/CD) 管道。

在这里，我们使用 sam pipeline init --bootstrap 命令为应用程序配置 CI/CD 部署管道。会 Amazon SAMCLI指导我们选择各个选项，并生成要与 CI/CD 系统结合使用的 Amazon 资源和配置文件。

```
[3] Reference application build resources
Enter the pipeline execution role ARN if you have previously created one, or we will create one for y
ou []:
Enter the CloudFormation execution role ARN if you have previously created one, or we will create one
 for you []:
Please enter the artifact bucket ARN for your Lambda function. If you do not have a bucket, we will c
reate one for you []:
Does your application contain any IMAGE type Lambda functions? [y/N]: n

[4] Summary
Below is the summary of the answers:
        1 - Account: 513423067560
        2 - Stage configuration name: dev
        3 - Region: us-west-2
        4 - Pipeline user: [to be created]
        5 - Pipeline execution role: [to be created]
        6 - CloudFormation execution role: [to be created]
        7 - Artifacts bucket: [to be created]
        8 - ECR image repository: [skipped]
Press enter to confirm the values above, or select an item to edit the value:

This will create the following required resources for the 'dev' configuration:
        - Pipeline IAM user
        - Pipeline execution role
        - CloudFormation execution role
        - Artifact bucket
Should we proceed with the creation? [y/N]:
```

有关更多详细信息，请参阅 使用 CI/CD 系统和管线进行部署。

对云端应用程序进行监控和问题排查

查看有关已部署资源的重要信息，收集日志，并使用 Amazon X-Ray等内置监控工具。

在这里，我们使用 sam list 命令查看已部署的资源。我们获取 API 端点并调用它，这会触发函数。然后，使用 sam logs 查看函数的日志。

```
➜  sam-app sam logs --stack-name sam-app
2023/03/13/[$LATEST]0a433e844dd445bd82d0d78cd55e0cdc 2023-03-13T21:06:42.075000 INIT_START Runtime Ve
rsion: python:3.9.v16   Runtime Version ARN: arn:aws:lambda:us-west-2::runtime:07a48df201798d627f2b95
0f03bb227aab4a655a1d019c3296406f95937e2525
2023/03/13/[$LATEST]0a433e844dd445bd82d0d78cd55e0cdc 2023-03-13T21:06:42.180000 START RequestId: 778e
4226-0a80-435f-929b-5b19292ed9a7 Version: $LATEST
2023/03/13/[$LATEST]0a433e844dd445bd82d0d78cd55e0cdc 2023-03-13T21:06:42.181000 END RequestId: 778e42
26-0a80-435f-929b-5b19292ed9a7
2023/03/13/[$LATEST]0a433e844dd445bd82d0d78cd55e0cdc 2023-03-13T21:06:42.182000 REPORT RequestId: 778
e4226-0a80-435f-929b-5b19292ed9a7       Duration: 1.69 ms       Billed Duration: 2 ms   Memory Size:
128 MB  Max Memory Used: 36 MB  Init Duration: 104.13 ms
```

有关更多详细信息，请参阅 监控应用程序。

在开发时将本地更改同步到云端

在本地计算机上开发时，自动将更改同步到云端。快速查看您所做的更改，并在云端执行测试和验证。

在这里，我们使用 sam sync --watch 命令来让 Amazon SAM CLI 监控本地更改。我们修改 `HelloWorldFunction` 代码，然后 Amazon SAM CLI 自动检测更改并将更新部署到云端。

```
-----------------------------------------------------------------------------------------------
Key                 HelloWorldFunctionIamRole
Description         Implicit IAM Role created for Hello World function
Value               arn:aws:iam::513423067560:role/sam-app-HelloWorldFunctionRole-15GLOUR9LMT1W

Key                 HelloWorldApi
Description         API Gateway endpoint URL for Prod stage for Hello World function
Value               https://ets1gv8lxi.execute-api.us-west-2.amazonaws.com/Prod/hello/

Key                 HelloWorldFunction
Description         Hello World Lambda Function ARN
Value               arn:aws:lambda:us-west-2:513423067560:function:sam-app-HelloWorldFunction-
yQDNe17r9maD
-----------------------------------------------------------------------------------------------

Stack update succeeded. Sync infra completed.

Infra sync completed.
CodeTrigger not created as CodeUri or DefinitionUri is missing for ServerlessRestApi.
Syncing Lambda Function HelloWorldFunction...
Manifest is not changed for (HelloWorldFunction), running incremental build
Building codeuri: /Users/evzz/Demo/what-is/sam-app/hello_world runtime: python3.9 metadata: {} archit
ecture: x86_64 functions: HelloWorldFunction
Running PythonPipBuilder:CopySource
Finished syncing Lambda Function HelloWorldFunction.
```

测试云端受支持的资源

调用事件并将事件传递给云端受支持的资源。

在这里，我们使用 sam remote invoke 命令测试部署在云端的 Lambda 函数。我们调用 Lambda 函数并接收其日志和响应。将 Lambda 函数配置为流式传输响应后， Amazon SAM CLI 会实时流式传回其响应。

```
→ lambda-streaming-app sam remote invoke StreamingFunction --stack-name lambda-streaming-app
```

了解更多

要进一步了解的更多信息 Amazon SAM，请参阅以下资源：

- 完整 Amazon SAM 研讨会-旨在指导您使用 Amazon SAM 提供的许多主要功能的研讨会。
- 使用 SAM 的会话 — 由 Amazon 无服务器开发人员支持团队制作的关于使用 Amazon SAM的视频系列。
- Serverless Land — 汇集了关于无 Amazon 服务器项目的最新信息、博客、视频、代码和学习资源的网站。

后续步骤

如果这是您第一次使用 Amazon SAM，请参阅入门 Amazon SAM。

的无服务器概念 Amazon Serverless Application Model

在使用 Amazon Serverless Application Model (Amazon SAM) 之前，请先了解基本的无服务器概念。

无服务器概念

事件驱动型架构

无服务器应用程序由单独的服务组成，例如 Amazon Lambda 用于计算的 Amazon 服务和用于数据库管理的 Amazon DynamoDB，每个服务都扮演一个专门的角色。然后，这些服务通过事件驱动型架构相互松散地集成。要了解有关事件驱动型架构的更多信息，请参阅什么是事件驱动型架构？。

基础设施即代码（IaC）

基础设施即代码（IaC）是一种以开发人员对待代码的方式对待基础设施的方法，将与应用程序代码开发相同的严密性应用于基础设施配置。您可以在模板文件中定义基础架构，将其部署到模板文件中 Amazon，然后为您 Amazon 创建资源。使用 IaC，您可以在代码中定义 Amazon 要配置的内容。有关更多信息，请参阅 Amazon Amazon 白皮书简介中的基础架构即代码。 DevOps

无服务器技术

借助 Amazon 无服务器技术，您无需管理自己的服务器即可构建和运行应用程序。所有服务器管理都是通过完成的 Amazon，它提供了许多好处，例如自动扩展和内置的高可用性，使您可以将自己的想法快速付诸实践。使用无服务器技术，您可以专注于产品的核心，而不必担心服务器的管理和操作。要了解有关无服务器的更多信息，请参阅以下内容：

- 无服务器开启 Amazon
- 《无服务器开发人员指南》：提供 Amazon 云中无服务器开发的概念性概述。

有关核心 Amazon 无服务器服务的基本介绍，请参阅 Serv erless 101：了解无服务器领域的无服务器服务。

无服务器应用程序

使用时 Amazon SAM，您可以在应用程序中管理相关资源，该应用程序由您的 Amazon SAM 项目和模板组成。您的应用程序中的所有资源都是在您的 Amazon SAM 模板中定义或引用的。在 Amazon SAM 处理您的模板时，它会创建 Amazon CloudFormation 资源。在中 Amazon CloudFormation，资源在称为堆栈的单个单元中进行管理，堆栈中的所有资源均由堆栈的 Amazon CloudFormation 模板定义。

入门 Amazon SAM

首先， Amazon SAM 请查看并完成本节中的主题。 Amazon SAM 先决条件提供了有关设置 Amazon 账户、创建 IAM 用户、创建密钥访问权限以及安装和配置账户的详细说明 Amazon SAM CLI。 完成先决条件后，你就可以准备好了，你可以在 Linux 安装 Amazon SAM CLI、Windows 和 macOS 操作系统上完成这项工作。安装完成后，你可以选择学习 Amazon SAM Hello World 教程。按照本教程将引导您完成使用 Amazon SAM创建基本无服务器应用程序的过程。完成本教程后，您就可以复习 如何使用 Amazon Serverless Application Model (Amazon SAM) 中详述的概念了。

主题

- Amazon SAM 先决条件
- 安装 Amazon SAM CLI
- 教程：使用以下命令部署 Hello World 应用程序 Amazon SAM

Amazon SAM 先决条件

在安装和使用 Amazon Serverless Application Model 命令行界面 (Amazon SAMCLI) 之前，请完成以下先决条件。

要使用 Amazon SAMCLI，您需要满足以下条件：

- Amazon 账户、 Amazon Identity and Access Management (IAM) 证书和 IAM 访问密钥对。
- 用于配置 Amazon 凭据的 Amazon Command Line Interface (Amazon CLI)。

主题

- 第 1 步：注册一个 Amazon 账户
- 第 2 步：创建 IAM 用户账户
- 第 3 步：创建访问密钥 ID 和秘密访问密钥
- 步骤 4：安装 Amazon CLI
- 步骤 5： Amazon CLI 使用配置 Amazon 凭证
- 后续步骤

第 1 步：注册一个 Amazon 账户

如果您没有 Amazon Web Services 账户，请完成以下步骤来创建一个。

要注册 Amazon Web Services 账户

1. 打开https://portal.aws.amazon.com/billing/注册。
2. 按照屏幕上的说明操作。

   在注册时，将接到电话或收到短信，要求使用电话键盘输入一个验证码。

   当您注册时 Amazon Web Services 账户，就会创建Amazon Web Services 账户根用户一个。根用户有权访问该账户中的所有 Amazon Web Services 服务 和资源。作为最佳安全实践，请为用户分配管理访问权限，并且只使用根用户来执行需要根用户访问权限的任务。

第 2 步：创建 IAM 用户账户

保护 IAM 用户

注册后 Amazon Web Services 账户，开启多重身份验证 (MFA)，保护您的管理用户。有关说明，请参阅《IAM 用户指南》中的 为 IAM 用户启用虚拟 MFA 设备（控制台）。

要允许其他用户访问您的 Amazon Web Services 账户 资源，请创建 IAM 用户。为了保护您的 IAM 用户，请启用 MFA 并仅向 IAM 用户授予执行任务所需的权限。

有关创建和保护 IAM 用户的更多信息，请参阅《IAM 用户指南》中的以下主题：

- 在你的 IAM 用户中创建 Amazon Web Services 账户
- 适用于 Amazon 资源的访问权限管理
- 基于 IAM 身份的策略示例

第 3 步：创建访问密钥 ID 和秘密访问密钥

要进行 CLI 访问，您需要访问密钥 ID 和秘密访问密钥。如果可能，请使用临时凭证代替长期访问密钥。临时凭证包括访问密钥 ID、秘密访问密钥，以及一个指示凭证何时到期的安全令牌。有关更多信息，请参阅 IAM 用户指南中的将临时证书与 Amazon 资源配合使用。

如果用户想在 Amazon 外部进行交互，则需要编程访问权限 Amazon Web Services Management Console。 Amazon APIs 和 Amazon Command Line Interface 需要访问密钥。可能的话，创建临时凭证，该凭证由一个访问密钥 ID、一个秘密访问密钥和一个指示凭证何时到期的安全令牌组成。

要向用户授予编程式访问权限，请选择以下选项之一。

哪个用户需要编程式访问权限？	目的	方式
IAM	使用短期证书签署对 Amazon CLI 或的编程请求 Amazon APIs（直接或使用 Amazon SDKs）。	按照 IAM 用户指南中的将临时证书与 Amazon 资源配合使用中的说明进行操作。
IAM	（不推荐使用） 使用长期证书签署对 Amazon CLI 或的编程请求 Amazon APIs（直接或使用 Amazon SDKs）。	按照《IAM 用户指南》中管理 IAM 用户的访问密钥中的说明进行操作。

步骤 4：安装 Amazon CLI

Amazon CLI 是一款开源工具，可让您 Amazon Web Services 服务 使用命令行外壳中的命令进行交互。 Amazon CLI 对于诸如配置凭据之类的活动， Amazon SAMCLI需要使用。要了解有关... 的更多信息 Amazon CLI，请参阅什么是 Amazon Command Line Interface？ 在《Amazon Command Line Interface 用户指南》中。

要安装 Amazon CLI，请参阅Amazon Command Line Interface 用户指南 Amazon CLI中的安装或更新最新版本的。

步骤 5： Amazon CLI 使用配置 Amazon 凭证

使用 IAM 身份中心配置证书

- 要使用 IAM Identity Center 配置证书，请参阅使用 Amazon 配置 SSO 向导配置您的个人资料。

要使用配置凭证 Amazon CLI

1. 从命令行处运行 `aws configure` 命令。
2. 进行以下配置。选择每个链接以了解更多信息：

   a. 访问密钥 ID
   b. 秘密访问密钥
   c. Amazon Web Services 区域
   d. 输出格式

   以下示例显示了示例值。

```
$ aws configure
AWS Access Key ID [None]: AKIAIOSFODNN7EXAMPLE
AWS Secret Access Key [None]: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
Default region name [None]: us-west-2
Default output format [None]: json
```

将此信息 Amazon CLI 存储在`credentials`和文件中命名的配置`config`文件（设置集合）`default`中。这些文件位于主目录下的 `.aws` 文件中。默认情况下，当您运行未明确指定要使用的配置文件的 Amazon CLI 命令时，将使用此配置文件中的信息。有关 `credentials` 文件的更多信息，请参阅《Amazon Command Line Interface 用户指南》中的配置和凭证文件设置。

有关配置凭证（例如使用现有配置和凭据文件）的更多信息，请参阅《Amazon Command Line Interface 用户指南》中的快速设置功能。

后续步骤

现在，您可以安装 Amazon SAMCLI并开始使用了 Amazon SAM。要安装 Amazon SAMCLI，请参阅安装 Amazon SAM CLI。

安装 Amazon SAM CLI

按照中的说明在支持的操作系统上安装最新版本的 Amazon Serverless Application Model 命令行界面 (Amazon SAMCLI) 步骤 4：安装 Amazon CLI。

有关管理，包括如何升级 Amazon SAMCLI、卸载或管理每夜构建版本，请参阅管理 Amazon SAM CLI 版本。

这是您第一次安装 Amazon SAM CLI？

在继续操作之前，请完成上一节中的所有先决条件。这包括：

1. 注册 Amazon 账户。
2. 设置安全访问权限 Amazon。
3. 创建访问密钥 ID 和秘密访问密钥。
4. 正在安装 Amazon CLI.
5. 配置 Amazon 凭证。

主题

- 安装 Amazon SAM CLI
- 排查 安装错误
- 后续步骤
- 可选：验证 Amazon SAMCLI 安装程序的完整性

安装 Amazon SAM CLI

Note

从 2023 年 9 月起， Amazon 将不再维护 Amazon SAMCLI (`aws/tap/aws-sam-cli`) 的 Amazon 托管Homebrew安装程序。如果您Homebrew使用安装和管理 Amazon SAMCLI，请参阅以下选项：

- 要继续使用 Homebrew，您可以使用社区托管的安装程序。有关更多信息，请参阅 管理 Amazon SAM CLI 替换为 Homebrew。
- 我们建议使用本页中记录的第一方安装方法之一。在使用其中一种方法之前，请参阅 切换自 Homebrew。
- 有关更多详细信息，请参阅 Release version: 1.121.0。

要安装 Amazon SAMCLI，请按照适合操作系统的说明进行操作。

Linux

x86_64 - command line installer

1. 将 Amazon SAM CLI .zip 文件下载到所选目录。
2. （可选）您可以在安装前验证安装程序的完整性。有关说明，请参阅可选：验证 Amazon SAMCLI 安装程序的完整性。
3. 将安装文件解压缩到您选择的目录中。以下是使用 `sam-installation` 子目录的示例。

> **Note**
>
> 如果您的操作系统没有内置的 unzip 命令，请使用等效命令。

```
$ unzip aws-sam-cli-linux-x86_64.zip -d sam-installation
```

4. 通过运行 `install` 可执行文件安装 Amazon SAM CLI。该可执行文件位于上一步骤中使用的目录中。以下是使用 `sam-installation` 子目录的示例：

```
$ sudo ./sam-installation/install
```

5. 验证安装。

```
$ sam --version
```

要确认安装成功，您应该会看到一个输出，该输出将以下括号内的文本替换为最新的可用版本：

```
SAM CLI, <latest version>
```

arm64 - command line installer

1. 将 Amazon SAM CLI .zip 文件下载到所选目录。
2. （可选）您可以在安装前验证安装程序的完整性。有关说明，请参阅可选：验证 Amazon SAMCLI 安装程序的完整性。
3. 将安装文件解压缩到您选择的目录中。以下是使用 `sam-installation` 子目录的示例。

> **Note**
>
> 如果您的操作系统没有内置的 unzip 命令，请使用等效命令。

```
$ unzip aws-sam-cli-linux-arm64.zip -d sam-installation
```

4. 通过运行 `install` 可执行文件安装 Amazon SAM CLI。该可执行文件位于上一步骤中使用的目录中。以下是使用 `sam-installation` 子目录的示例：

```
$ sudo ./sam-installation/install
```

5. 验证安装。

```
$ sam --version
```

要确认安装成功，您应该会看到如下所示的输出，但其中括号内的文本将替换为最新的 SAM CLI 版本：

```
SAM CLI, <latest version>
```

macOS

安装步骤

使用软件包安装程序安装 Amazon SAMCLI。此外，软件包安装程序提供两种安装方法让您选择：GUI 和命令行。您可以为所有用户安装，也可以只为当前用户安装。要为所有用户安装，需要获得超级用户授权。

GUI - All users

下载软件包安装程序并安装 Amazon SAMCLI

> **Note**
>
> 如果您之前通过 Homebrew 或 pip 安装了 Amazon SAM CLI，则需要先将其卸载。有关说明，请参阅正在卸载 Amazon SAM CLI。

1. 将 macOS pkg 下载到您选择的目录：
   - 对于运行英特尔处理器的 Mac，请选择 x86 _64 —-x86_64.pkg aws-sam-cli-macos
   - 对于运行 Apple 硅芯片的 Mac，选择 arm64 —-arm64.pkg aws-sam-cli-macos

   > **Note**
   >
   > （可选）您可以选择在安装前验证安装程序的完整性。有关说明，请参阅可选：验证 Amazon SAMCLI 安装程序的完整性。

2. 运行您下载的文件，然后按照屏幕上的说明继续完成简介、自述和许可步骤。
3. 在目标选择中，选择为这台计算机的所有用户安装。
4. 在安装类型中，选择 Amazon SAM CLI 要安装的位置，然后按下安装。推荐的默认位置是 `/usr/local/aws-sam-cli`。

   > **Note**
   >
   > 要使用 sam 命令调用 Amazon SAM CLI，安装程序会自动在 `/usr/local/bin/sam` 和 `/usr/local/aws-sam-cli/sam` 或您选择的安装文件夹之间创建符号链接。

5. Amazon SAM CLI 将进行安装，系统将显示安装成功消息。按下关闭。

验证安装是否成功

- 通过运行以下命令验证 Amazon SAM CLI 是否已正确安装以及符号链接是否已配置：

```
$ which sam
/usr/local/bin/sam
$ sam --version
SAM CLI, <latest version>
```

GUI - Current user

下载并安装 Amazon SAMCLI

> **Note**
> 如果您之前通过 Homebrew 或 pip 安装了 Amazon SAM CLI，则需要先将其卸载。有关说明，请参阅正在卸载 Amazon SAM CLI。

1. 将 macOS pkg 下载到您选择的目录：
   - 对于运行英特尔处理器的 Mac，请选择 x86 _64 —-x86_64.pkg aws-sam-cli-macos
   - 对于运行 Apple 硅芯片的 Mac，选择 arm64 —-arm64.pkg aws-sam-cli-macos

   > **Note**
   > （可选）您可以选择在安装前验证安装程序的完整性。有关说明，请参阅可选：验证 Amazon SAMCLI 安装程序的完整性。

2. 运行您下载的文件，然后按照屏幕上的说明继续完成简介、自述和许可步骤。
3. 对于目标选择，选择仅为我安装。如果没有看到此选项，请转到下一步。
4. 对于安装类型，请执行以下操作：
   1. 选择 Amazon SAM CLI 要安装的位置。默认位置是 `/usr/local/aws-sam-cli`。选择您拥有写入权限的位置。要更改安装位置，请选择本地并选择您的位置。完成后按下继续。
   2. 如果您在上一步中没有看到仅为我安装的选项，请选择更改安装位置 > 仅为我安装，然后按下继续。
   3. 按下安装。
5. Amazon SAM CLI 将进行安装，系统将显示安装成功消息。按下关闭。

要创建符号链接

- 要使用 sam 命令调用 Amazon SAM CLI，您必须在 Amazon SAM CLI 程序和您的 $PATH 之间手动创建符号链接。通过修改并运行以下命令来创建符号链接：

```
$ sudo ln -s /path-to/aws-sam-cli/sam /path-to-symlink-directory/sam
```

- *sudo*— 如果您的用户具有写入权限$PATH，则sudo不是必需的。否则，sudo 是必需的。
- *path-to*— Amazon SAMCLI 程序安装位置的路径。例如 /Users/myUser/Desktop。
- *path-to-symlink-directory*— 您的$PATH环境变量。默认位置是 /usr/local/bin。

验证安装是否成功

- 通过运行以下命令验证 Amazon SAM CLI 是否已正确安装以及符号链接是否已配置：

```
$ which sam
/usr/local/bin/sam
$ sam --version
SAM CLI, <latest version>
```

Command line - All users

下载并安装 Amazon SAMCLI

> **Note**
> 如果您之前通过 Homebrew 或 pip 安装了 Amazon SAM CLI，则需要先将其卸载。有关说明，请参阅正在卸载 Amazon SAM CLI。

1. 将 macOS pkg 下载到您选择的目录：
   - 对于运行英特尔处理器的 Mac，请选择 x86 _64 —-x86_64.pkg aws-sam-cli-macos
   - 对于运行 Apple 硅芯片的 Mac，选择 arm64 —-arm64.pkg aws-sam-cli-macos

   > **Note**
   > （可选）您可以选择在安装前验证安装程序的完整性。有关说明，请参阅可选：验证 Amazon SAMCLI 安装程序的完整性。

2. 修改并运行安装脚本：

```
$ sudo installer -pkg path-to-pkg-installer/name-of-pkg-installer -target /
installer: Package name is AWS SAM CLI
installer: Upgrading at base path /
installer: The upgrade was successful.
```

> **Note**
>
> 要使用 sam 命令调用 Amazon SAM CLI，安装程序会自动在 `/usr/local/bin/sam` 和 `/usr/local/aws-sam-cli/sam` 之间创建符号链接。

验证安装是否成功

- 通过运行以下命令验证 Amazon SAM CLI 是否已正确安装以及符号链接是否已配置：

```
$ which sam
/usr/local/bin/sam
$ sam --version
SAM CLI, <latest version>
```

Command line - Current user

下载并安装 Amazon SAMCLI

> **Note**
>
> 如果您之前通过 Homebrew 或 pip 安装了 Amazon SAM CLI，则需要先将其卸载。有关说明，请参阅正在卸载 Amazon SAM CLI。

1. 将 macOS pkg 下载到您选择的目录：
   - 对于运行英特尔处理器的 Mac，请选择 x86 _64 —-x86_64.pkg aws-sam-cli-macos
   - 对于运行 Apple 硅芯片的 Mac，选择 arm64 —-arm64.pkg aws-sam-cli-macos

> **Note**
>
> （可选）您可以选择在安装前验证安装程序的完整性。有关说明，请参阅可选：验证 Amazon SAMCLI 安装程序的完整性。

2. 确定您有写入权限的安装目录。然后，使用模板创建 xml 文件并对其进行修改以反映您的安装目录。目录必须已经存在。

   例如，如果*path-to-my-directory*替换为/Users/myUser/Desktop，则aws-sam-cli程序文件夹将安装在那里。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/
PropertyList-1.0.dtd">
<plist version="1.0">
  <array>
    <dict>
      <key>choiceAttribute</key>
      <string>customLocation</string>
      <key>attributeSetting</key>
      <string>path-to-my-directory</string>
      <key>choiceIdentifier</key>
      <string>default</string>
    </dict>
  </array>
</plist>
```

3. 保存 xml 文件并通过运行以下命令验证其是否有效：

```
$ installer -pkg path-to-pkg-installer \
-target CurrentUserHomeDirectory \
-showChoicesAfterApplyingChangesXML path-to-your-xml-file
```

   输出应显示将应用于 Amazon SAM CLI 程序的首选项。

4. 运行以下命令来安装 Amazon SAMCLI：

```
$ installer -pkg path-to-pkg-installer \
-target CurrentUserHomeDirectory \
-applyChoiceChangesXML path-to-your-xml-file
```

```
# Example output
installer: Package name is AWS SAM CLI
installer: choices changes file 'path-to-your-xml-file' applied
installer: Upgrading at base path base-path-of-xml-file
installer: The upgrade was successful.
```

要创建符号链接

- 要使用 sam 命令调用 Amazon SAM CLI，您必须在 Amazon SAM CLI 程序和您的 $PATH 之间手动创建符号链接。通过修改并运行以下命令来创建符号链接：

```
$ sudo ln -s /path-to/aws-sam-cli/sam /path-to-symlink-directory/sam
```

  - *sudo*— 如果您的用户具有写入权限$PATH，则sudo不是必需的。否则，sudo 是必需的。
  - *path-to*— Amazon SAMCLI 程序安装位置的路径。例如 /Users/myUser/Desktop。
  - *path-to-symlink-directory*— 您的$PATH环境变量。默认位置是 /usr/local/bin。

验证安装是否成功

- 通过运行以下命令验证 Amazon SAM CLI 是否已正确安装以及符号链接是否已配置：

```
$ which sam
/usr/local/bin/sam
$ sam --version
SAM CLI, <latest version>
```

Windows

Windows 安装程序 (MSI) 文件是 Windows 操作系统的软件包安装程序文件。

按照以下步骤使用 MSI 文件安装 Amazon SAM CLI。

1. 下载 Amazon SAM CLI 64 位版本。
2. （可选）您可以在安装前验证安装程序的完整性。有关说明，请参阅可选：验证 Amazon SAMCLI 安装程序的完整性。

3. 验证安装。

   安装完成后，打开新的命令提示符或 PowerShell 提示符进行验证。您应该能够从命令行调用 `sam`。

   ```
   sam --version
   ```

   成功安装后 Amazon SAMCLI，您应该会看到类似下面的输出：

   ```
   SAM CLI, <latest version>
   ```

4. 启用长路径（仅限 Windows 10 及更高版本）。

   > **Important**
   >
   > Amazon SAMCLI可能会与超过 Windows 最大路径限制的文件路径进行交互。由于 Windows 10 的 MAX_PATH 限制，这可能会在运行 `sam init` 时导致错误。若要解决此问题，必须配置新的长路径行为。

   要启用长路径，请参阅 Microsoft Windows 应用程序开发文档中的在 Windows 10 版本 1607 及更高版本中启用长路径。

5. 安装 Git。

   要使用 `sam init` 命令下载示例应用程序，还必须安装 Git。有关说明，请参见安装 Git。

排查 安装错误

Linux

Docker 错误："无法连接到 Docker 进程守护程序。Docker 进程守护程序是否在此主机上运行？"

在某些情况下，要为 `ec2-user` 提供访问 Docker 进程守护程序的权限，您可能需要重新启动实例。如果您收到此错误，请尝试重启实例。

Shell 错误："找不到命令"

如果您收到此错误，则表示 Shell 无法在路径中找到 Amazon SAM CLI 可执行文件。验证 Amazon SAM CLI 可执行文件安装目录的位置，然后验证该目录是否位于路径中。

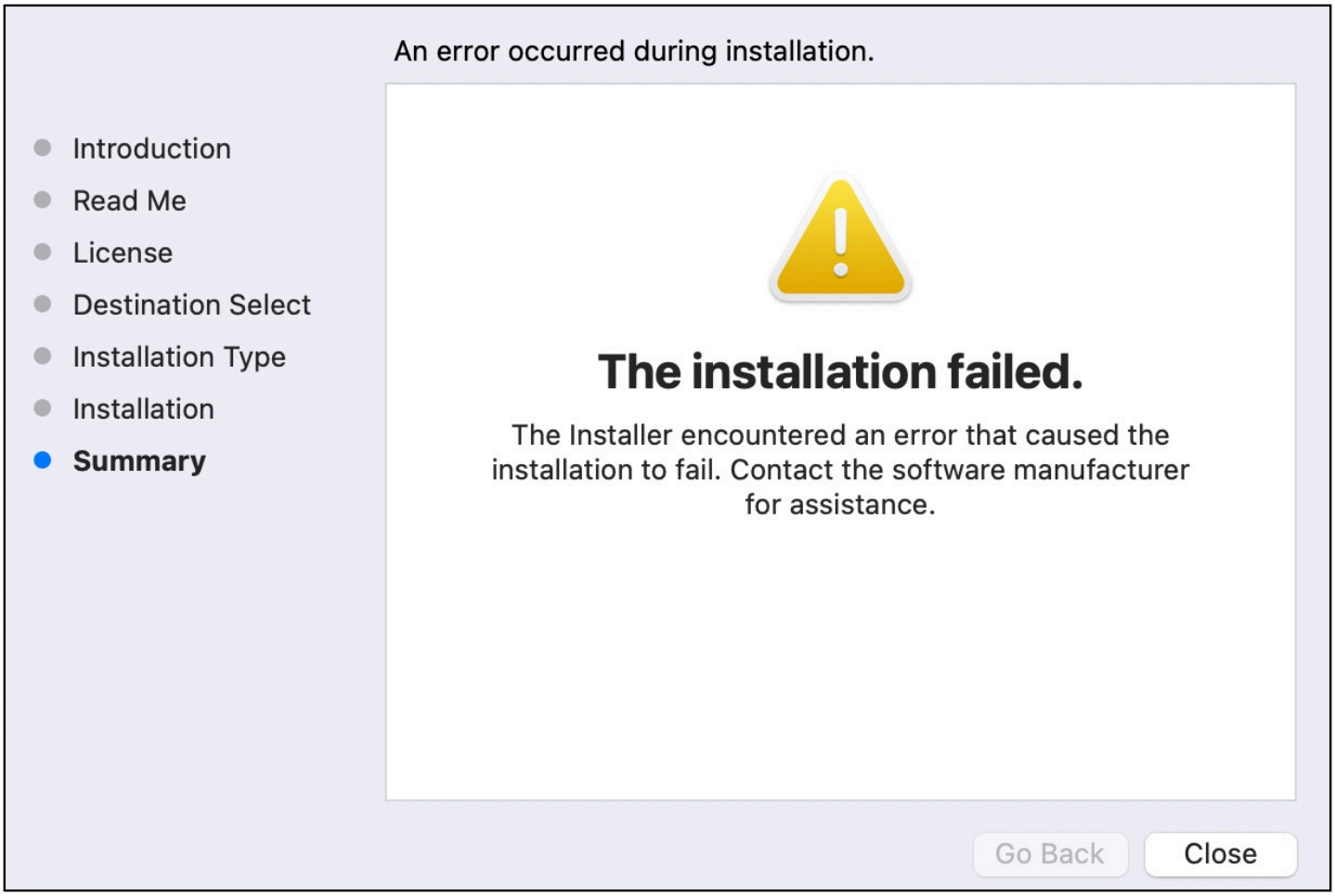
Amazon SAMCLI错误："/lib64/libc.so.6：找不到版本 `GLIBC_2.14' (/.so.1 需要该版本) " usr/local/aws-sam-cli/dist/libz

如果您收到此错误，则表示您使用的是不受支持的 Linux 版本，并且内置的 glibc 版本已过时。请尝试以下任一方法：

- 将您的 Linux 主机升级到最新的 64 位 CentOS、Fedora、Ubuntu 或 Amazon Linux 2 发行版。
- 按照 安装 Amazon SAM CLI 的说明进行操作。

macOS

安装失败



如果您是在为用户安装 Amazon SAM CLI，并且选择了您没有写入权限的安装目录，则可能会出现此错误。请尝试以下任一方法：

1. 选择您具有写入权限的其他安装目录。

2. 删除安装程序。然后重新下载并运行安装程序。

后续步骤

要了解有关 Amazon SAM CLI 的更多信息并开始构建自己的无服务器应用程序，请参阅以下内容：

- 教程：使用以下命令部署 Hello World 应用程序 Amazon SAM— 下载、构建和部署基本无服务器应用程序的 Step-by-step说明。
- 完整 Amazon SAM 研讨会-旨在指导您使用 Amazon SAM 提供的许多主要功能的研讨会。
- Amazon SAM 示例应用程序和模式-来自社区作者的示例应用程序和模式，可供您进一步试用。

可选：验证 Amazon SAMCLI 安装程序的完整性

使用软件包安装程序安装 Amazon Serverless Application Model 命令行界面 (Amazon SAMCLI) 时，可以在安装前验证安装程序的完整性。此步骤是可选的，但我们强烈建议您执行此步骤。

可供您使用的两个验证选项是：

- 验证软件包安装程序签名文件。
- 验证软件包安装程序的哈希值。

如果可用于您的平台，我们建议您验证签名文件选项。此选项提供额外的安全层，因为密钥值在此处发布，并且与 GitHub 存储库分开管理。

主题

- 验证安装程序包签名文件
- 验证哈希值

验证安装程序包签名文件

Linux

arm64：命令行安装程序

Amazon SAM 使用 GnuPG 对.zip 安装程序进行签名 Amazon SAMCLI。验证过程需要执行以下步骤：

1. 使用主公钥验证签署人公钥。
2. 使用签署人公钥验证 Amazon SAM CLI 软件包安装程序。

验证签署人公钥的完整性

1. 复制主公钥并将其作为 `.txt` 文件保存到本地计算机。例如 *primary-public-key.txt*。

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v2.0.22 (GNU/Linux)

mQINBGRuSzMBEADsqiwOy78w7F4+sshaMFRIwRGNRm94p5Qey2KMZBxekFtoryVD
D9jEOnvupx4tvhfBHz5EcUHCEOdl4MTqdBy6vVAshozgxVb9RE8JpECn5lw7XC69
4Y7Gy1TKKQMEWtDXElkGxIFdUWvWjSnPlzfnoXwQYGeE93CUS3h5dImP22Yk1Ct6
eGGhlcbg1X4L8EpFMj7GvcsU8f7ziVI/PyC1Xwy39Q8/I67ip5eU5ddxO/xHqrbL
YC7+8pJPbRMej2twT2LrcpWWYAbprMtRoa6WfE0/thoo3xhHpIMHdPfAA86ZNGIN
kRLjGUg7jnPTRW4Oin3pCc8nT4Tfc1QERkHm641gTC/jUvpmQsM6h/FUVP2i5iE/
JHpJcMuL2Mg6zDo3x+3gTCf+Wqz3rZzxB+wQT3yryZs6efcQy7nROiRxYBxCSXX0
2cNYzsYLb/bYaW8yqWIHD5IqKhw269gp2E5Khs60zgS3CorMb5/xHgXjUCVgcu8a
a8ncdf9fjl3WS5p0ohetPbO2ZjWv+MaqrZOmUIgKbA4RpWZ/fU97P5BW9ylwmIDB
sWy0cMxg8MlvSdLytPieogaM0qMg3u5qXRGBr6Wmevkty0qgnmpGGc5zPiUbtOE8
CnFFqyxBpj5IOnG0KZGVihvn+iRxrv6GO7WWO92+Dc6m94U0EEiBR7QiOwARAQAB
tDRBV1MgU0FNIENMSSBQcmltYXJ5IDxhd3Mtc2FtLWNsaS1wcmltYXJ5QGFtYXpv
bi5jb20+iQI/BBMBCQApBQJkbkszAhsvBQkHhM4ABwsJCAcDAgEGFQgCCQoLBBYC
AwECHgECF4AACgkQQv1fenOtiFqTuhAAzi5+ju5UVOWqHKevOJSO08T4QB8HcqAE
SVO3mY6/j29knkcL8ubZP/DbpV7QpHPI2PB5qSXsiDTP3IYPbeY78zHSDjljaIK3
njJLMScFeGPyfPpwMsuY4nzrRIgAtXShPA8N/k4ZJcafnpNqKj7QnPxiC1KaIQWm
pOtvb8msUF3/s0UTa5Ys/lNRhVC0eGg32ogXGdojZA2kHZWdm9udLo4CDrDcrQT7
NtDcJASapXSQL63XfAS3snEc4e1941YxcjfYZ33rel8K9juyDZfi1slWR/L3AviI
QFIaqSHzyOtP1oinUkoVwL8ThevKD3Ag9CZflZLzNCV7yqlF8RlhEZ4zcE/3s9El
WzCFsozb5HfE1AZonmrDh3SyOEIBMcS6vG5dWnvJrAuSYv2rX38++K5Pr/MIAfOX
DOI1rtA+XDsHNv9lSwSy0lt+iClawZANO9IXCiN1rOYcVQlwzDFwCNWDgkwdOqS0
gOA2f8NF9lE5nBbeEuYquoOl1Vy8+ICbgOFs9LoWZlnVh7/RyY6ssowiU9vGUnHI
L8f9jqRspIz/Fm3JD86ntZxLVGkeZUz62FqErdohYfkFIVcv7GONTEyrz5HLlnpv
FJ0MR0HjrMrZrnOVZnwBKhpbLocTsH+3t5It4ReYEX0f1DIOL/KRwPvjMvBVkXY5
hblRVDQoOWc=
=d9oG
-----END PGP PUBLIC KEY BLOCK-----
```

2. 将主公钥导入到您的密钥环中。

```
$ gpg --import primary-public-key.txt
```

```
gpg: directory `/home/.../.gnupg' created
gpg: new configuration file `/home/.../.gnupg/gpg.conf' created
gpg: WARNING: options in `/home/.../.gnupg/gpg.conf' are not yet active during this
 run
gpg: keyring `/home/.../.gnupg/secring.gpg' created
gpg: keyring `/home/.../.gnupg/pubring.gpg' created
gpg: /home/.../.gnupg/trustdb.gpg: trustdb created
gpg: key 73AD885A: public key "Amazon SAM CLI Primary <aws-sam-cli-
primary@amazon.com>" imported
gpg: Total number processed: 1
gpg:               imported: 1  (RSA: 1)
```

3. 复制签署人公钥并将其作为 `.txt` 文件保存到本地计算机。例如 *signer-public-key.txt*。


```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v2.0.22 (GNU/Linux)

mQINBGgrxIgBEADGCTudveeeVbWpZDGX9Ni57mBRMVSJwQJ6F/PC34jw0DozxTtd
H+ZPsXLvLwerN/DVXbK8E1qNZ5RGptak8j7MPz+MC3n4txibEJpB61vpjJJM+9cC
7whaMLDT/SbykHYXdrnHqa8KsUJl7rPLJcaRN722NSxvYVMIOA9ffVXV7cfEyZi5
MbYF2Gc9LNbKaknImIva7EKeeh2/wI6YCqC5yytyfWU5dL6oHXsgTnFL9mhziMxv
WhyzawyJG6EJZsJ3WLlbIKApN6XZSXyCxOvlBrebYZjD5v0nA+TJaQ7is8atjtOI
DGe0AViw7kO8ChTpjA7YG/Uu7n/Fy7qLF/3Nz0b6cBNjemjBazQ3A3KNCpi5hqFM
Uo1WpoVLr5CXQnc0B3fBUnTIoxi0Sk5MKjH9AbYxfgqEX0ZJB9hAlc6LIEy0Yru6
MMBrIHE86IMl1NfE/DeLnCdPG23+1PttwyOt3+9z5QwmPe3VPpEfCySPcdxHKZSP
rLile8qDznEvlPDvQ0qkBxdMtVa2yct5VJkdqy6UrN2xa0dpspHjRUjHh/EY/xMt
fwMUjOKohaZ/1pjotCcksAsZWUxCNcFvLYxuxeytVk4F09Es1hj4ihhLUI+43/ic
3DHSEiext7Q8/UccNArkhSCT7UOvvL7QTuP+pjYTyiC8Vx6g/Y5Ht5+qywARAQAB
tDBBV1MgU0FNIENMSSBUZWFtIDxhd3Mtc2FtLWNsaS1zaWduZXJAYW1hem9uLmNv
bT6JAj8EEwEJACkFAmgrxIgCGy8FCQPCZwAHCwkIBwMCAQYVCAIJCgsEFgIDAQIe
AQIXgAAKCRBAlKuxvt/atJo6EAC/5C8uJs76W5f5V5XNAMzwBFiZuYpop3DRReCo
P68ZZylokAC9ShRZnIOujpDJtlNS7T/G00BzmcpspkYYE531ALaXcHWmb9XV0Ajg
J8iboAVBLY0C7mhL/cbJ3v9QlpXXjyTuhexkJCV8rdHVX/0H8WqTZplEaRuZ7p8q
PMxddg4ClwstYuH3O/dmNdlGqfb4Fqy8MnV1yGSXRs5Jf+sDlN2UO4mbpyk/mr1c
f/jFxmx86IkCWJVvdXWCVTe2AFy3NHCdLtdnEvFhokCOQd9wibUWX0j9vq4cVRZT
qamnpAQaOlH3lXOwrjqo8b1AIPoRWSfMtCYvh6kA8MAJv4cAznzXILSLtOE0mzaU
qp5qoy37wNIjeztX6c/q4wss05qTlJhnNu4s3nh5VHultooaYpmDxp+ala5TWeuM
KZDI4KdAGF4z0Raif+N53ndOYIiXkY0goUbsPCnVrCwoK9PjjyoJncq7c14wNl5O
IQUZEjyYAQDGZqs5XSfY4zW2cCXatrfozKF7R1kSU14DfJwPUyksoNAQEQezfXyq
kr0gfIWK1r2nMdqS7WgSx/ypS5kdyrHuPZdaYfEVtuezpoT2lQQxOSZqqlp5hI4R
nqmPte53WXJhbC0tgTIJWn+Uy/d5Q/aSIfD6o8gNLS1BDs1j1ku0XKu1sFCHUcZG
aerdsIkCHAQQAQkABgUCaCvFeAAKCRBC/V96c62IWt3/D/9gOLzWtz62lqJRCsri
wcA/yz88ayKb/GUv3FCT5Nd9JZt8y1tW+AE3SPTdcpfZmt5UN2sRzljO61mpKJzp
eBvYQ9og/34ZrRQqeg8bz02u34LKYl1gD0xY0bWtB7TGIxIZZYqZECoPR0Dp6ZzB
```

```
abzkRSsJkEk0vbZzJhfWFYs98qfp/G0suFSBE79O8Am33DB2jQ/Sollh1VmNE6Sv
EOgR6+2yEkS2D0+msJMa/V82v9gBTPnxSlNV1d8Dduvt9rbM3LoxiNXUgx/s52yY
U6H3bwUcQ3UY6uRe1UWo5QnMFcDwfg43+q5rmjB4xQyX/BaQyF5K0hZyG+42/pH1
EMwl8qN617FTxo3hvQUi/cBahlhQ8EVYsGnHDVxLCisbq5iZvp7+XtmMy1Q417gT
EQRo8feJh31elGWlccVR2pZgIm1PQ69dzzseHnnKkGhifik0bDGo5/IH2EgI1KFn
SG399RMU/qRzOPLVP3i+zSJmhMqG8cnZaUwE5V4P21vQSclhhd2Hv/C4SVKNqA2i
+oZbHj2vAkuzTTL075AoANebEjPGqwsKZi5mWUE5Pa931JeiXxWZlEB7rkgQ1PAB
fsDBhYLt4MxCWAhifLMA6uQ4BhXu2RuXOqNfSbqa8jVF6DB6cD8eAHGpPKfJOl30
LtZnq+n4SfeNbZjD2FQWZR4CrA==
=lHfs
-----END PGP PUBLIC KEY BLOCK-----
```

4. 将签署人公钥导入到您的密钥环中。

```
$ gpg --import signer-public-key.txt

gpg: key 4094ABB1BEDFDAB4: public key "Amazon SAM CLI Team <aws-sam-cli-
signer@amazon.com>" imported
gpg: Total number processed: 1
gpg:               imported: 1  (RSA: 1)
gpg: no ultimately trusted keys found
```

记下输出中的密钥值。例如 *4094ABB1BEDFDAB4*。

5. 使用密钥值获取并验证签署人公钥的指纹。

```
$ gpg --fingerprint 4094ABB1BEDFDAB4

pub   rsa4096 2025-05-19 [SCEA] [expires: 2027-05-19]
      EF46 3E86 CA31 933B B688  CC1A 4094 ABB1 BEDF DAB4
uid           [ unknown] Amazon SAM CLI Team <aws-sam-cli-signer@amazon.com>
```

指纹应与以下内容匹配：

```
EF46 3E86 CA31 933B B688  CC1A 4094 ABB1 BEDF DAB4
```

如果指纹字符串不匹配，请不要使用 Amazon SAM CLI 安装程序。通过在aws-sam-cli GitHub 存储库中创建问题来上报给 Amazon SAM 团队。

6. 验证签署人公钥的签名：

```
$ gpg --check-sigs 4094ABB1BEDFDAB4
```

```
pub   rsa4096 2025-05-19 [SCEA] [expires: 2027-05-19]
      EF463E86CA31933BB688CC1A4094ABB1BEDFDAB4
uid           [ unknown] Amazon SAM CLI Team <aws-sam-cli-signer@amazon.com>
sig!3        4094ABB1BEDFDAB4 2025-05-19  [self-signature]
sig!         42FD5F7A73AD885A 2025-05-19 Amazon SAM CLI Primary <aws-sam-cli-
primary@amazon.com>
```

如果看到 `1 signature not checked due to a missing key`，请重复前面的步骤，将主公钥和签署人公钥导入到密钥环中。

您应该会看到列出的主公钥和签署人公钥的密钥值。

现在，您已经验证了签署人公钥的完整性，您可以使用签署人公钥来验证 Amazon SAM CLI 软件包安装程序。

验证 Amazon SAM CLI 软件包安装程序的完整性

1. 获取 Amazon SAM CLI 软件包签名文件 - 使用以下命令下载 Amazon SAM CLI 软件包安装程序的签名文件：

```
$ wget https://github.com/aws/aws-sam-cli/releases/latest/download/aws-sam-cli-
linux-arm64.zip.sig
```

2. 验证签名文件 – 将两个下载 `.sig` 和 `.zip` 文件作为参数传递给 gpg 命令。以下是示例：

```
$ gpg --verify aws-sam-cli-linux-arm64.zip.sig aws-sam-cli-linux-arm64.zip
```

该输出值应该类似于以下内容：

```
gpg: Signature made Mon 19 May 2025 01:21:57 AM UTC using RSA key ID
 4094ABB1BEDFDAB4
gpg: Good signature from "Amazon SAM CLI Team <aws-sam-cli-signer@amazon.com>"
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: EF46 3E86 CA31 933B B688  CC1A 4094 ABB1 BEDF DAB4
```

- 可以忽略消息`WARNING: This key is not certified with a trusted signature!`。之所以出现此警告，是因为您的个人 PGP 密钥（如果您有）和 Amazon SAM CLI 密钥之间没有信任链。有关更多信息，请参阅信任 Web。

- 如果输出包含短语 `BAD signature`，则检查是否正确执行了此过程。如果您继续收到此回复，请通过在aws-sam-cli GitHub 存储库中创建问题来上报给 Amazon SAM 团队，并避免使用下载的文件。

消息`Good signature from "Amazon SAM CLI Team <aws-sam-cli-signer@amazon.com>"`表示签名已通过验证，您可以继续安装。

x86_64 - 命令行安装程序

Amazon SAM 使用 GnuPG 对.zip 安装程序进行签名 Amazon SAMCLI。验证过程需要执行以下步骤：

1. 使用主公钥验证签署人公钥。
2. 使用签署人公钥验证 Amazon SAM CLI 软件包安装程序。

验证签署人公钥的完整性

1. 复制主公钥并将其作为 `.txt` 文件保存到本地计算机。例如 *`primary-public-key.txt`*。

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v2.0.22 (GNU/Linux)

mQINBGRuSzMBEADsqiwOy78w7F4+sshaMFRIwRGNRm94p5Qey2KMZBxekFtoryVD
D9jEOnvupx4tvhfBHz5EcUHCEOdl4MTqdBy6vVAshozgxVb9RE8JpECn5lw7XC69
4Y7Gy1TKKQMEWtDXElkGxIFdUWvWjSnPlzfnoXwQYGeE93CUS3h5dImP22Yk1Ct6
eGGhlcbg1X4L8EpFMj7GvcsU8f7ziVI/PyC1Xwy39Q8/I67ip5eU5ddxO/xHqrbL
YC7+8pJPbRMej2twT2LrcpWWYAbprMtRoa6WfE0/thoo3xhHpIMHdPfAA86ZNGIN
kRLjGUg7jnPTRW4Oin3pCc8nT4Tfc1QERkHm641gTC/jUvpmQsM6h/FUVP2i5iE/
JHpJcMuL2Mg6zDo3x+3gTCf+Wqz3rZzxB+wQT3yryZs6efcQy7nROiRxYBxCSXX0
2cNYzsYLb/bYaW8yqWIHD5IqKhw269gp2E5Khs60zgS3CorMb5/xHgXjUCVgcu8a
a8ncdf9fjl3WS5p0ohetPbO2ZjWv+MaqrZOmUIgKbA4RpWZ/fU97P5BW9ylwmIDB
sWy0cMxg8MlvSdLytPieogaM0qMg3u5qXRGBr6Wmevkty0qgnmpGGc5zPiUbtOE8
CnFFqyxBpj5IOnG0KZGVihvn+iRxrv6GO7WWO92+Dc6m94U0EEiBR7QiOwARAQAB
tDRBV1MgU0FNIENMSSBQcmltYXJ5IDxhd3Mtc2FtLWNsaS1wcmltYXJ5QGFtYXpv
bi5jb20+iQI/BBMBCQApBQJkbkszAhsvBQkHhM4ABwsJCAcDAgEGFQgCCQoLBBYC
AwECHgECF4AACgkQQv1fenOtiFqTuhAAzi5+ju5UVOWqHKevOJSO08T4QB8HcqAE
SVO3mY6/j29knkcL8ubZP/DbpV7QpHPI2PB5qSXsiDTP3IYPbeY78zHSDjljaIK3
njJLMScFeGPyfPpwMsuY4nzrRIgAtXShPA8N/k4ZJcafnpNqKj7QnPxiC1KaIQWm
pOtvb8msUF3/s0UTa5Ys/lNRhVC0eGg32ogXGdojZA2kHZWdm9udLo4CDrDcrQT7
NtDcJASapXSQL63XfAS3snEc4e1941YxcjfYZ33rel8K9juyDZfi1slWR/L3AviI
```

```
QFIaqSHzyOtP1oinUkoVwL8ThevKD3Ag9CZflZLzNCV7yqlF8RlhEZ4zcE/3s9El
WzCFsozb5HfE1AZonmrDh3SyOEIBMcS6vG5dWnvJrAuSYv2rX38++K5Pr/MIAfOX
DOI1rtA+XDsHNv9lSwSy0lt+iClawZANO9IXCiN1rOYcVQlwzDFwCNWDgkwdOqS0
gOA2f8NF9lE5nBbeEuYquoOl1Vy8+ICbgOFs9LoWZlnVh7/RyY6ssowiU9vGUnHI
L8f9jqRspIz/Fm3JD86ntZxLVGkeZUz62FqErdohYfkFIVcv7GONTEyrz5HLlnpv
FJ0MR0HjrMrZrnOVZnwBKhpbLocTsH+3t5It4ReYEX0f1DIOL/KRwPvjMvBVkXY5
hblRVDQoOWc=
=d9oG
-----END PGP PUBLIC KEY BLOCK-----
```

2. 将主公钥导入到您的密钥环中。

```
$ gpg --import primary-public-key.txt

gpg: directory `/home/.../.gnupg' created
gpg: new configuration file `/home/.../.gnupg/gpg.conf' created
gpg: WARNING: options in `/home/.../.gnupg/gpg.conf' are not yet active during this
 run
gpg: keyring `/home/.../.gnupg/secring.gpg' created
gpg: keyring `/home/.../.gnupg/pubring.gpg' created
gpg: /home/.../.gnupg/trustdb.gpg: trustdb created
gpg: key 73AD885A: public key "Amazon SAM CLI Primary <aws-sam-cli-
primary@amazon.com>" imported
gpg: Total number processed: 1
gpg:               imported: 1  (RSA: 1)
```

3. 复制签署人公钥并将其作为 `.txt` 文件保存到本地计算机。例如 *signer-public-key.txt*。

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v2.0.22 (GNU/Linux)

mQINBGgrxIgBEADGCTudveeeVbWpZDGX9Ni57mBRMVSJwQJ6F/PC34jw0DozxTtd
H+ZPsXLvLwerN/DVXbK8E1qNZ5RGptak8j7MPz+MC3n4txibEJpB61vpjJJM+9cC
7whaMLDT/SbykHYXdrnHqa8KsUJl7rPLJcaRN722NSxvYVMIOA9ffVXV7cfEyZi5
MbYF2Gc9LNbKaknImIva7EKeeh2/wI6YCqC5yytyfWU5dL6oHXsgTnFL9mhziMxv
WhyzawyJG6EJZsJ3WLlbIKApN6XZSXyCxOvlBrebYZjD5v0nA+TJaQ7is8atjtOI
DGe0AViw7kO8ChTpjA7YG/Uu7n/Fy7qLF/3Nz0b6cBNjemjBazQ3A3KNCpi5hqFM
Uo1WpoVLr5CXQnc0B3fBUnTIoxi0Sk5MKjH9AbYxfgqEX0ZJB9hAlc6LIEy0Yru6
MMBrIHE86IMl1NfE/DeLnCdPG23+1PttwyOt3+9z5QwmPe3VPpEfCySPcdxHKZSP
rLile8qDznEvlPDvQ0qkBxdMtVa2yct5VJkdqy6UrN2xa0dpspHjRUjHh/EY/xMt
fwMUjOKohaZ/1pjotCcksAsZWUxCNcFvLYxuxeytVk4F09Es1hj4ihhLUI+43/ic
3DHSEiext7Q8/UccNArkhSCT7UOvvL7QTuP+pjYTyiC8Vx6g/Y5Ht5+qywARAQAB
tDBBV1MgU0FNIENMSSBUZWFtIDxhd3Mtc2FtLWNsaS1zaWduZXJAYW1hem9uLmNv
```

```
bT6JAj8EEwEJACkFAmgrxIgCGy8FCQPCZwAHCwkIBwMCAQYVCAIJCgsEFgIDAQIe
AQIXgAAKCRBAlKuxvt/atJo6EAC/5C8uJs76W5f5V5XNAMzwBFiZuYpop3DRReCo
P68ZZylokAC9ShRZnIOujpDJtlNS7T/G00BzmcpspkYYE531ALaXcHWmb9XV0Ajg
J8iboAVBLY0C7mhL/cbJ3v9QlpXXjyTuhexkJCV8rdHVX/0H8WqTZplEaRuZ7p8q
PMxddg4ClwstYuH3O/dmNdlGqfb4Fqy8MnV1yGSXRs5Jf+sDlN2UO4mbpyk/mr1c
f/jFxmx86IkCWJVvdXWCVTe2AFy3NHCdLtdnEvFhokCOQd9wibUWX0j9vq4cVRZT
qamnpAQaOlH3lXOwrjqo8b1AIPoRWSfMtCYvh6kA8MAJv4cAznzXILSLtOE0mzaU
qp5qoy37wNIjeztX6c/q4wss05qTlJhnNu4s3nh5VHultooaYpmDxp+ala5TWeuM
KZDI4KdAGF4z0Raif+N53ndOYIiXkY0goUbsPCnVrCwoK9PjjyoJncq7c14wNl5O
IQUZEjyYAQDGZqs5XSfY4zW2cCXatrfozKF7R1kSU14DfJwPUyksoNAQEQezfXyq
kr0gfIWK1r2nMdqS7WgSx/ypS5kdyrHuPZdaYfEVtuezpoT2lQQxOSZqqlp5hI4R
nqmPte53WXJhbC0tgTIJWn+Uy/d5Q/aSIfD6o8gNLS1BDs1j1ku0XKu1sFCHUcZG
aerdsIkCHAQQAQkABgUCaCvFeAAKCRBC/V96c62IWt3/D/9gOLzWtz62lqJRCsri
wcA/yz88ayKb/GUv3FCT5Nd9JZt8y1tW+AE3SPTdcpfZmt5UN2sRzljO61mpKJzp
eBvYQ9og/34ZrRQqeg8bz02u34LKYl1gD0xY0bWtB7TGIxIZZYqZECoPR0Dp6ZzB
abzkRSsJkEk0vbZzJhfWFYs98qfp/G0suFSBE79O8Am33DB2jQ/Sollh1VmNE6Sv
EOgR6+2yEkS2D0+msJMa/V82v9gBTPnxSlNV1d8Dduvt9rbM3LoxiNXUgx/s52yY
U6H3bwUcQ3UY6uRe1UWo5QnMFcDwfg43+q5rmjB4xQyX/BaQyF5K0hZyG+42/pH1
EMwl8qN617FTxo3hvQUi/cBahlhQ8EVYsGnHDVxLCisbq5iZvp7+XtmMy1Q417gT
EQRo8feJh31elGWlccVR2pZgIm1PQ69dzzseHnnKkGhifik0bDGo5/IH2EgI1KFn
SG399RMU/qRzOPLVP3i+zSJmhMqG8cnZaUwE5V4P21vQSclhhd2Hv/C4SVKNqA2i
+oZbHj2vAkuzTTL075AoANebEjPGqwsKZi5mWUE5Pa931JeiXxWZlEB7rkgQ1PAB
fsDBhYLt4MxCWAhifLMA6uQ4BhXu2RuXOqNfSbqa8jVF6DB6cD8eAHGpPKfJOl30
LtZnq+n4SfeNbZjD2FQWZR4CrA==
=lHfs
-----END PGP PUBLIC KEY BLOCK-----
```


4. 将签署人公钥导入到您的密钥环中。

```
$ gpg --import signer-public-key.txt

gpg: key 4094ABB1BEDFDAB4: public key "Amazon SAM CLI Team <aws-sam-cli-
signer@amazon.com>" imported
gpg: Total number processed: 1
gpg:               imported: 1  (RSA: 1)
gpg: no ultimately trusted keys found
```

记下输出中的密钥值。例如 *4094ABB1BEDFDAB4*。

5. 使用密钥值获取并验证签署人公钥的指纹。

```
$ gpg --fingerprint 4094ABB1BEDFDAB4
```

```
pub   rsa4096 2025-05-19 [SCEA] [expires: 2027-05-19]
      EF46 3E86 CA31 933B B688  CC1A 4094 ABB1 BEDF DAB4
uid           [ unknown] Amazon SAM CLI Team <aws-sam-cli-signer@amazon.com>
```

指纹应与以下内容匹配：

```
EF46 3E86 CA31 933B B688  CC1A 4094 ABB1 BEDF DAB4
```

如果指纹字符串不匹配，请不要使用 Amazon SAM CLI 安装程序。通过在aws-sam-cli GitHub 存储库中创建问题来上报给 Amazon SAM 团队。

6. 验证签署人公钥的签名：

```
$ gpg --check-sigs 4094ABB1BEDFDAB4

pub   rsa4096 2025-05-19 [SCEA] [expires: 2027-05-19]
      EF463E86CA31933BB688CC1A4094ABB1BEDFDAB4
uid           [ unknown] Amazon SAM CLI Team <aws-sam-cli-signer@amazon.com>
sig!3        4094ABB1BEDFDAB4 2025-05-19  [self-signature]
sig!         42FD5F7A73AD885A 2025-05-19 Amazon SAM CLI Primary <aws-sam-cli-
primary@amazon.com>
```

如果看到 `1 signature not checked due to a missing key`，请重复前面的步骤，将主公钥和签署人公钥导入到密钥环中。

您应该会看到列出的主公钥和签署人公钥的密钥值。

现在，您已经验证了签署人公钥的完整性，您可以使用签署人公钥来验证 Amazon SAM CLI 软件包安装程序。

验证 Amazon SAM CLI 软件包安装程序的完整性

1. 获取 Amazon SAM CLI 软件包签名文件 - 使用以下命令下载 Amazon SAM CLI 软件包安装程序的签名文件：

```
$ wget https://github.com/aws/aws-sam-cli/releases/latest/download/aws-sam-cli-
linux-x86_64.zip.sig
```

2. 验证签名文件 – 将两个下载 `.sig` 和 `.zip` 文件作为参数传递给 gpg 命令。以下是示例：

```
$ gpg --verify aws-sam-cli-linux-x86_64.zip.sig aws-sam-cli-linux-x86_64.zip
```

该输出值应该类似于以下内容：

```
gpg: Signature made Mon 19 May 2025 01:21:57 AM UTC using RSA key ID
 4094ABB1BEDFDAB4
gpg: Good signature from "Amazon SAM CLI Team <aws-sam-cli-signer@amazon.com>"
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: EF46 3E86 CA31 933B B688  CC1A 4094 ABB1 BEDF DAB4
```

- 可以忽略消息`WARNING: This key is not certified with a trusted signature!`。之所以出现此警告，是因为您的个人 PGP 密钥（如果您有）和 Amazon SAM CLI 密钥之间没有信任链。有关更多信息，请参阅信任 Web。
- 如果输出包含短语 `BAD signature`，则检查是否正确执行了此过程。如果您继续收到此回复，请通过在aws-sam-cli GitHub 存储库中创建问题来上报给 Amazon SAM 团队，并避免使用下载的文件。

消息`Good signature from "Amazon SAM CLI Team <aws-sam-cli-signer@amazon.com>"`表示签名已通过验证，您可以继续安装。

macOS

GUI 和命令行安装程序

您可以使用 `pkgutil` 工具或手动验证 Amazon SAM CLI 软件包安装程序签名文件的完整性。

使用 pkgutil 进行验证

1. 运行以下命令，提供下载的安装程序在本地计算机上的路径：

   ```
   $ pkgutil --check-signature /path/to/aws-sam-cli-installer.pkg
   ```

   以下是示例：

   ```
   $ pkgutil --check-signature /Users/user/Downloads/aws-sam-cli-macos-arm64.pkg
   ```

2. 从输出中找到 Developer ID Installer: AMZN Mobile LLC 的 SHA256 fingerprint。以下是示例：

```
Package "aws-sam-cli-macos-arm64.pkg":
   Status: signed by a developer certificate issued by Apple for distribution
   Notarization: trusted by the Apple notary service
   Signed with a trusted timestamp on: 2023-05-16 20:29:29 +0000
   Certificate Chain:
    1. Developer ID Installer: AMZN Mobile LLC (94KV3E626L)
       Expires: 2027-06-28 22:57:06 +0000
       SHA256 Fingerprint:
           49 68 39 4A BA 83 3B F0 CC 5E 98 3B E7 C1 72 AC 85 97 65 18 B9 4C
           BA 34 62 BF E9 23 76 98 C5 DA
       ------------------------------------------------------------------------
    2. Developer ID Certification Authority
       Expires: 2031-09-17 00:00:00 +0000
       SHA256 Fingerprint:
           F1 6C D3 C5 4C 7F 83 CE A4 BF 1A 3E 6A 08 19 C8 AA A8 E4 A1 52 8F
           D1 44 71 5F 35 06 43 D2 DF 3A
       ------------------------------------------------------------------------
    3. Apple Root CA
       Expires: 2035-02-09 21:40:36 +0000
       SHA256 Fingerprint:
           B0 B1 73 0E CB C7 FF 45 05 14 2C 49 F1 29 5E 6E DA 6B CA ED 7E 2C
           68 C5 BE 91 B5 A1 10 01 F0 24
```

3. Developer ID Installer: AMZN Mobile LLC SHA256 fingerprint 应匹配以下值：

```
49 68 39 4A BA 83 3B F0 CC 5E 98 3B E7 C1 72 AC 85 97 65 18 B9 4C BA 34 62 BF E9 23
 76 98 C5 DA
```

如果指纹字符串不匹配，请不要使用 Amazon SAM CLI 安装程序。通过在aws-sam-cli GitHub 存储库中创建问题来上报给 Amazon SAM 团队。如果指纹字符串匹配，您可以继续使用软件包安装程序。

手动验证软件包安装程序

- 请参阅 Apple 支持网站上的如何验证手动下载的 Apple 软件更新的真实性。

Windows

Amazon SAMCLI安装程序打包为Windows操作系统的MSI文件。

验证安装程序的完整性

1. 右键单击安装程序，打开属性窗口。
2. 选择数字签名选项卡。
3. 在签名列表中，选择 Amazon Web Services, Inc.，然后选择详细信息。
4. 选择常规选项卡 (如果尚未选择)，然后选择查看证书。
5. 选择详细信息选项卡，然后选择显示下拉列表中的全部 (如果尚未选择)。
6. 向下滚动直至您看到指纹字段，然后选择指纹。这将在下部窗口中显示整个指纹值。
7. 将指纹值与以下值进行匹配。如果值匹配，就继续安装。如果不是，请通过在aws-sam-cli GitHub 存储库中创建问题来上报给 Amazon SAM 团队。

```
d52eb68bffe6ae165b3b05c3e1f9cc66da7eeac0
```

验证哈希值

Linux

x86_64 - 命令行安装程序

使用以下命令生成哈希值，验证下载的安装程序文件的完整性和真实性：

```
$ sha256sum aws-sam-cli-linux-x86_64.zip
```

输出应与以下示例类似：

```
<64-character SHA256 hash value> aws-sam-cli-linux-x86_64.zip
```

将包含 64 个字符的 SHA-256 哈希值与您所需的 Amazon SAM CLI 版本在 GitHub 上的Amazon SAM CLI 发布说明中的哈希值进行比较。

macOS

GUI 和命令行安装程序

使用以下命令生成哈希值，验证下载的安装程序的完整性和真实性：

```
$ shasum -a 256 path-to-pkg-installer/name-of-pkg-installer

# Examples
```

```
$ shasum -a 256 ~/Downloads/aws-sam-cli-macos-arm64.pkg
$ shasum -a 256 ~/Downloads/aws-sam-cli-macos-x86_64.pkg
```

将包含 64 个字符的 SHA-256 哈希值与 GitHub 存储库上的 Amazon SAM CLI 发布说明中相应的值进行比较。

教程：使用以下命令部署 Hello World 应用程序 Amazon SAM

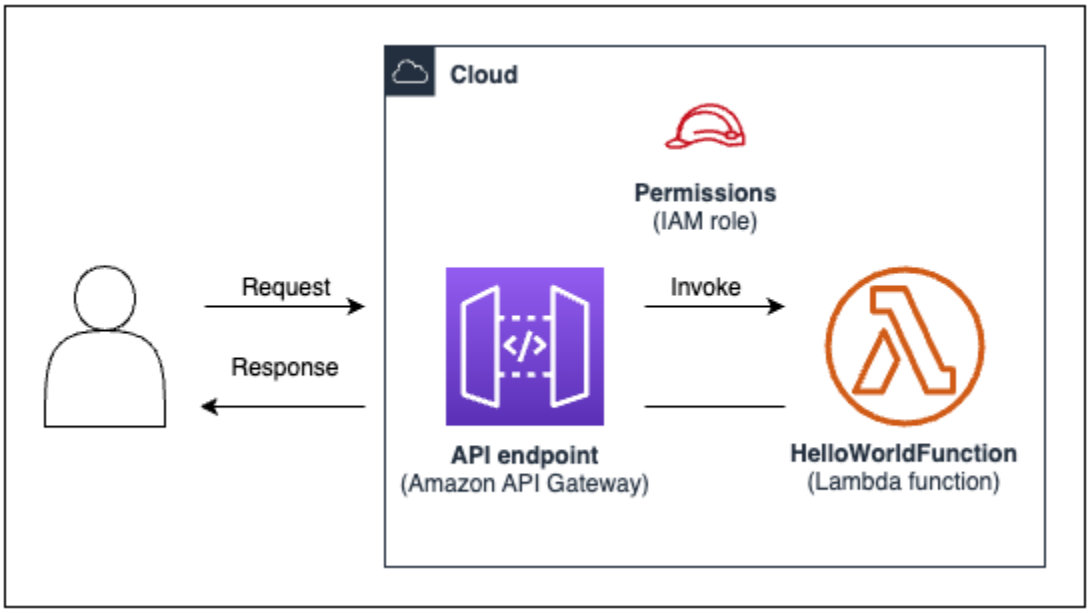
在本教程中，您将使用 Amazon Serverless Application Model 命令行界面 (Amazon SAM CLI) 来完成以下操作：

- 初始化、构建和部署示例 Hello World 应用程序。
- 进行本地更改并同步到 Amazon CloudFormation。
- 在开发主机上执行本地测试。
- 从 Amazon Web Services 云中删除示例应用程序。

示例 Hello World 应用程序实现了基本的 API 后端。它由以下资源组成：

- Amazon API Gateway – 将用于调用函数的 API 端点。
- Amazon Lambda– 处理 HTTP API GET 请求并返回 `hello world` 消息的函数。
- Amazon Identity and Access Management (IAM) 角色 — 为服务提供安全交互的权限。

下图显示此应用程序的组件：



主题

- 先决条件
- 第 1 步：初始化示例 Hello World 应用程序
- 第 2 步：构建应用程序
- 步骤 3：将您的应用程序部署到 Amazon Web Services 云
- 第 4 步：运行应用程序
- 第 5 步：在中与你的函数进行交互 Amazon Web Services 云
- 步骤 6：修改您的应用程序并将其同步到 Amazon Web Services 云
- 第 7 步：（可选）在本地测试应用程序
- 第 8 步：从中删除您的应用程序 Amazon Web Services 云
- 故障排除
- 了解更多

先决条件

确认您是否已完成以下操作：

- Amazon SAM 先决条件
- 安装 Amazon SAM CLI

第 1 步：初始化示例 Hello World 应用程序

在此步骤中，您将使用 Amazon SAM CLI 在您的本地计算机上创建示例 Hello World 应用程序项目。

初始化示例 Hello World 应用程序

1. 在命令行中，从您选择的起始目录运行以下内容：

```
$ sam init
```

> **Note**
>
> 此命令会初始化您的无服务器应用程序，从而创建您的项目目录。此目录将包含多个文件和文件夹。最重要的文件是 `template.yaml`。这是你的 Amazon SAM 模板。您的

python 版本必须与 sam init 命令创建的 `template.yaml` 文件中列出的 python 版本匹配。

2. 的 Amazon SAM CLI 将指导您完成新应用程序的初始化。配置以下内容：

   1. 选择Amazon 快速入门模板以选择起始模板。
   2. 选择 Hello World 示例模板并下载。
   3. 使用 Python 运行时和`zip`包类型。
   4. 在本教程中，请选择退出 Amazon X-Ray 跟踪。要了解更多信息，请参阅什么是 Amazon X-Ray？ 在《Amazon X-Ray 开发人员指南》中。
   5. 在本教程中，请选择退出使用 Amazon CloudWatch 应用程序见解进行监控。要了解更多信息，请参阅《亚马逊 CloudWatch 用户指南》中的 “亚马逊 CloudWatch 应用程序见解”。
   6. 在本教程中，请选择不在 Lambda 函数上设置 JSON 格式的结构化日志记录。
   7. 将应用程序命名为 sam-app。

   要使用 Amazon SAM CLI 互动流程：

   - 方括号 (`[ ]`) 表示默认值。将答案留空以选择默认值。
   - 输入 **y** 以选择是，输入 **n** 以选择否。

   以下是 `sam init` 交互式流程的示例：

```
$ sam init
...
Which template source would you like to use?
    1 - AWS Quick Start Templates
    2 - Custom Template Location
Choice: 1

Choose an AWS Quick Start application template
    1 - Hello World Example
    2 - Data processing
    3 - Hello World Example with Powertools for AWS Lambda
    4 - Multi-step workflow
    5 - Scheduled task
    6 - Standalone function
    7 - Serverless API
    8 - Infrastructure event management
```

```
    9 - Lambda Response Streaming
   10 - GraphQLApi Hello World Example
   11 - Full Stack
   12 - Lambda EFS example
   13 - Serverless Connector Hello World Example
   14 - Multi-step workflow with Connectors
   15 - DynamoDB Example
   16 - Machine Learning
Template: 1

Use the most popular runtime and package type? (Python and zip) [y/N]: y

Would you like to enable X-Ray tracing on the function(s) in your application?  [y/
N]: ENTER

Would you like to enable monitoring using CloudWatch Application Insights?
For more info, please view https://docs.aws.amazon.com/AmazonCloudWatch/latest/
monitoring/cloudwatch-application-insights.html [y/N]: ENTER

Would you like to set Structured Logging in JSON format on your Lambda functions?
 [y/N]: ENTER

Project name [sam-app]: ENTER
```

3. 的 Amazon SAM CLI 下载您的起始模板并在本地计算机上创建应用程序项目目录结构。以下是示例 Amazon SAM CLI 输出：

```
Cloning from https://github.com/aws/aws-sam-cli-app-templates (process may take a
 moment)

    -----------------------
    Generating application:
    -----------------------
    Name: sam-app
    Runtime: python3.9
    Architectures: x86_64
    Dependency Manager: pip
    Application Template: hello-world
    Output Directory: .
    Configuration file: sam-app/samconfig.toml

    Next steps can be found in the README file at sam-app/README.md
```

```
Commands you can use next
=========================
[*] Create pipeline: cd sam-app && sam pipeline init --bootstrap
[*] Validate SAM template: cd sam-app && sam validate
[*] Test Function in the Cloud: cd sam-app && sam sync --stack-name {stack-name} --
watch
```

4. 在命令行中，移至新创建的 `sam-app` 目录。以下是一个示例 Amazon SAM CLI 已创建：

> **Note**
>
> 如果`tree`命令未自动安装 auto，请运行以下命令：`brew install tree`

```
$ cd sam-app

$ tree

### README.md
### __init__.py
### events
#   ### event.json
### hello_world
#   ### __init__.py
#   ### app.py
#   ### requirements.txt
### samconfig.toml
### template.yaml
### tests
    ### __init__.py
    ### integration
    #   ### __init__.py
    #   ### test_api_gateway.py
    ### requirements.txt
    ### unit
        ### __init__.py
        ### test_handler.py

6 directories, 14 files
```

需要重点介绍的一些重要文件：

- `hello_world/app.py` – 包含您的 Lambda 函数代码。
- `hello_world/requirements.txt`— 包含任何 Python 您的 Lambda 函数所需的依赖关系。
- `samconfig.toml`— 用于存储应用程序使用的默认参数的配置文件 Amazon SAM CLI.
- `template.yaml`— 包含您的应用程序基础架构代码的 Amazon SAM 模板。

现在，您的本地计算机上有了编写完整的无服务器应用程序！

第 2 步：构建应用程序

在此步骤中，您将使用 Amazon SAM CLI 来构建您的应用程序并为部署做准备。当你构建时，Amazon SAM CLI 创建一个`.aws-sam`目录并在其中整理函数依赖关系、项目代码和项目文件。

要构建应用程序

- 从 `sam-app` 项目目录，在命令行中运行以下命令：

```
$ sam build
```

> **Note**
> 如果你没有 Python 在您的本地计算机上，改用该sam build --use-container 命令。的 Amazon SAM CLI 将创建一个 Docker 包含函数运行时和依赖项的容器。此命令需要 Docker 在你的本地计算机上。要安装 Docker，请参阅 安装 Docker。

以下是示例 Amazon SAM CLI 输出：

```
$ sam build
Starting Build use cache
Manifest file is changed (new hash: 3298f1304...d4d421) or dependency folder (.aws-
sam/deps/4d3dfad6-a267-47a6-a6cd-e07d6fae318c) is missing for (HelloWorldFunction),
 downloading dependencies and copying/building source
Building codeuri: /Users/.../Demo/sam-tutorial1/sam-app/hello_world runtime:
 python3.9 metadata: {} architecture: x86_64 functions: HelloWorldFunction
Running PythonPipBuilder:CleanUp
Running PythonPipBuilder:ResolveDependencies
Running PythonPipBuilder:CopySource
Running PythonPipBuilder:CopySource
```

```
Build Succeeded

Built Artifacts  : .aws-sam/build
Built Template   : .aws-sam/build/template.yaml

Commands you can use next
=========================
[*] Validate SAM template: sam validate
[*] Invoke Function: sam local invoke
[*] Test Function in the Cloud: sam sync --stack-name {{stack-name}} --watch
[*] Deploy: sam deploy --guided
```

以下是 Amazon SAM CLI 创建的 `.aws-sam` 目录的简短示例：

```
.aws-sam
### build
#   ### HelloWorldFunction
#   #   ### __init__.py
#   #   ### app.py
#   #   ### requirements.txt
#   ### template.yaml
### build.toml
```

需要重点介绍的一些重要文件：

- `build/HelloWorldFunction` – 包含您的 Lambda 函数代码和依赖项。这些区域有： Amazon SAM CLI 为应用程序中的每个函数创建一个目录。
- `build/template.yaml`— 包含部署 Amazon CloudFormation 时引用的 Amazon SAM 模板副本。
- `build.toml`— 存储引用的默认参数值的配置文件 Amazon SAM CLI 在构建和部署应用程序时。

您现在已准备好将应用程序部署到 Amazon Web Services 云。

步骤 3：将您的应用程序部署到 Amazon Web Services 云

> **Note**
>
> 此步骤需要配置 Amazon 凭证。有关更多信息，请参阅Amazon SAM 先决条件中的步骤 5：Amazon CLI 使用配置 Amazon 凭证。

在此步骤中，您将使用 Amazon SAM CLI 将您的应用程序部署到 Amazon Web Services 云。的 Amazon SAM CLI 将执行以下操作：

- 指导您配置应用程序设置以进行部署。
- 将应用程序文件上载到 Amazon Simple Storage Service (Amazon S3)。
- 将您的 Amazon SAM 模板转换为 Amazon CloudFormation 模板。然后，它会将您的模板上传到 Amazon CloudFormation 服务以配置您的 Amazon 资源。

部署 应用程序

1. 从 `sam-app` 项目目录，在命令行中运行以下命令：

```
$ sam deploy --guided
```

2. 关注 Amazon SAM CLI 用于配置应用程序设置的交互式流程。配置以下内容：

   1. Amazon CloudFormation 堆栈名称 — 堆栈是您可以作为一个单元管理的 Amazon 资源集合。要了解更多信息，请参阅《Amazon CloudFormation 用户指南》中的使用堆栈。
   2. Amazon Web Services 区域要将您的 Amazon CloudFormation 堆栈部署到的。有关更多信息，请参阅《Amazon CloudFormation 用户指南》中的 Amazon CloudFormation 端点。
   3. 在本教程中，请取消选择在部署前确认更改。
   4. 允许创建 IAM 角色 — 这允许您创 Amazon SAM 建 API Gateway 资源和 Lambda 函数资源进行交互所必需的 IAM 角色。
   5. 在本教程中，请取消选择禁用回滚。
   6. HelloWorldFunction 未定义授权即允许 — 显示此消息是因为您的 API Gateway 端点已配置为无需授权即可公开访问。由于这是你的 Hello World 应用程序的预期配置，请允许 Amazon SAM CLI 继续。有关配置授权的更多信息，请参阅 使用您的 Amazon SAM 模板控制 API 访问权限。

7. 将参数保存到配置文件 – 这将使用您的部署首选项更新应用程序的 `samconfig.toml` 文件。
8. 选择默认配置文件名称。
9. 选择默认配置环境。

下面是一个 `sam deploy --guided` 交互式流程的示例输出：

```
$ sam deploy --guided

Configuring SAM deploy
======================

    Looking for config file [samconfig.toml] :  Found
    Reading default arguments  :  Success

    Setting default arguments for 'sam deploy'
    =========================================
    Stack Name [sam-app]: ENTER
    AWS Region [us-west-2]: ENTER
    #Shows you resources changes to be deployed and require a 'Y' to initiate
 deploy
    Confirm changes before deploy [Y/n]: n
    #SAM needs permission to be able to create roles to connect to the resources in
 your template
    Allow SAM CLI IAM role creation [Y/n]: ENTER
    #Preserves the state of previously provisioned resources when an operation
 fails
    Disable rollback [y/N]: ENTER
    HelloWorldFunction may not have authorization defined, Is this okay? [y/N]: y
    Save arguments to configuration file [Y/n]: ENTER
    SAM configuration file [samconfig.toml]: ENTER
    SAM configuration environment [default]: ENTER
```

3. 的 Amazon SAM CLI 通过执行以下操作来部署您的应用程序：
   - 的 Amazon SAM CLI 创建一个 Amazon S3 存储桶并上传您的`.aws-sam`目录。
   - 的 Amazon SAM CLI 将您的 Amazon SAM 模板转换为 Amazon CloudFormation 服务并将其上传到服务。 Amazon CloudFormation
   - Amazon CloudFormation 配置您的资源。

在部署期间， Amazon SAM CLI 显示您的进度。下面是一个示例输出：

```
Looking for resources needed for deployment:

    Managed S3 bucket: aws-sam-cli-managed-default-samclisam-s3-demo-
bucket-1a4x26zbcdkqr
    A different default S3 bucket can be set in samconfig.toml

    Parameter "stack_name=sam-app" in [default.deploy.parameters] is defined as a
 global parameter [default.global.parameters].
    This parameter will be only saved under [default.global.parameters] in /
Users/.../Demo/sam-tutorial1/sam-app/samconfig.toml.

    Saved arguments to config file
    Running 'sam deploy' for future deployments will use the parameters saved
 above.
    The above parameters can be changed by modifying samconfig.toml
    Learn more about samconfig.toml syntax at
    https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/
serverless-sam-cli-config.html

File with same data already exists at sam-app/da3c598813f1c2151579b73ad788cac8,
 skipping upload

    Deploying with following values
    ===============================
    Stack name                   : sam-app
    Region                       : us-west-2
    Confirm changeset            : False
    Disable rollback             : False
    Deployment s3 bucket         : aws-sam-cli-managed-default-samclisam-s3-demo-
bucket-1a4x26zbcdkqr
    Capabilities                 : ["CAPABILITY_IAM"]
    Parameter overrides          : {}
    Signing Profiles             : {}

Initiating deployment
=====================

File with same data already exists at sam-
app/2bebf67c79f6a743cc5312f6dfc1efee.template, skipping upload


Waiting for changeset to be created..
```

```
CloudFormation stack changeset
-------------------------------------------------------------------------------------------
Operation                           LogicalResourceId
 ResourceType                        Replacement
-------------------------------------------------------------------------------------------
* Modify                             HelloWorldFunction
 AWS::Lambda::Function               False
* Modify                             ServerlessRestApi
 AWS::ApiGateway::RestApi            False
- Delete                             AwsSamAutoDependencyLayerNestedSt
 AWS::CloudFormation::Stack          N/A
                                     ack
-------------------------------------------------------------------------------------------


Changeset created successfully. arn:aws:cloudformation:us-
west-2:012345678910:changeSet/samcli-deploy1678917603/22e05525-08f9-4c52-a2c4-
f7f1fd055072


2023-03-15 12:00:16 - Waiting for stack create/update to complete

CloudFormation events from stack operations (refresh every 0.5 seconds)
-------------------------------------------------------------------------------------------
ResourceStatus                      ResourceType
 LogicalResourceId                   ResourceStatusReason
-------------------------------------------------------------------------------------------
UPDATE_IN_PROGRESS                  AWS::Lambda::Function
 HelloWorldFunction                  -
UPDATE_COMPLETE                     AWS::Lambda::Function
 HelloWorldFunction                  -
UPDATE_COMPLETE_CLEANUP_IN_PROGRE   AWS::CloudFormation::Stack          sam-app
                          -
SS
DELETE_IN_PROGRESS                  AWS::CloudFormation::Stack
 AwsSamAutoDependencyLayerNestedSt   -
                                                                        ack
DELETE_COMPLETE                     AWS::CloudFormation::Stack
 AwsSamAutoDependencyLayerNestedSt   -
                                                                        ack
UPDATE_COMPLETE                     AWS::CloudFormation::Stack          sam-app
                          -
-------------------------------------------------------------------------------------------
```

```
CloudFormation outputs from deployed stack
------------------------------------------------------------------------------------------
Outputs
------------------------------------------------------------------------------------------
Key                 HelloWorldFunctionIamRole
Description         Implicit IAM Role created for Hello World function
Value               arn:aws:iam::012345678910:role/sam-app-
HelloWorldFunctionRole-15GLOUR9LMT1W

Key                 HelloWorldApi
Description         API Gateway endpoint URL for Prod stage for Hello World
 function
Value               https://<restapiid>.execute-api.us-west-2.amazonaws.com/Prod/
hello/

Key                 HelloWorldFunction
Description         Hello World Lambda Function ARN
Value               arn:aws:lambda:us-west-2:012345678910:function:sam-app-
HelloWorldFunction-yQDNe17r9maD
------------------------------------------------------------------------------------------



Successfully created/updated stack - sam-app in us-west-2
```

您的应用程序现已部署并在 Amazon Web Services 云! 中运行

第 4 步：运行应用程序

在此步骤中，您将向 API 端点发送 GET 请求并查看 Lambda 函数输出。

获取您的 API 端点值

1. 根据显示的信息 Amazon SAM CLI 在上一步中，找到相应`Outputs`部分。在该部分找到您的 `HelloWorldApi` 资源以查找 HTTP 端点值。下面是一个示例输出：

```
------------------------------------------------------------------------------------------
Outputs
------------------------------------------------------------------------------------------
...
Key                 HelloWorldApi
Description         API Gateway endpoint URL for Prod stage for Hello World
 function
```

```
Value               https://ets1gv8lxi.execute-api.us-west-2.amazonaws.com/Prod/
hello/
...
-----------------------------------------------------------------------------------------------
```

2. 或者，您可以使用 sam list endpoints --output json 命令获取此信息。下面是一个示例输出：

```
$ sam list endpoints --output json
2023-03-15 12:39:19 Loading policies from IAM...
2023-03-15 12:39:25 Finished loading policies from IAM.
[
  {
    "LogicalResourceId": "HelloWorldFunction",
    "PhysicalResourceId": "sam-app-HelloWorldFunction-yQDNe17r9maD",
    "CloudEndpoint": "-",
    "Methods": "-"
  },
  {
    "LogicalResourceId": "ServerlessRestApi",
    "PhysicalResourceId": "ets1gv8lxi",
    "CloudEndpoint": [
      "https://ets1gv8lxi.execute-api.us-west-2.amazonaws.com/Prod",
      "https://ets1gv8lxi.execute-api.us-west-2.amazonaws.com/Stage"
    ],
    "Methods": [
      "/hello['get']"
    ]
  }
]
```

要调用函数

- 使用浏览器或命令行向 API 端点发送 GET 请求。以下是使用 curl 命令的示例。

```
$ curl https://ets1gv8lxi.execute-api.us-west-2.amazonaws.com/Prod/hello/
{"message": "hello world"}
```

第 5 步：在中与你的函数进行交互 Amazon Web Services 云

在此步骤中，您将使用 Amazon SAM CLI 在中调用您的 Lambda 函数。 Amazon Web Services 云

要调用云端的 Lambda 函数

1. 记下上一步中函数的 `LogicalResourceId`。它应该是 `HelloWorldFunction`。
2. 从 `sam-app` 项目目录，在命令行中运行以下命令：

```
$ sam remote invoke HelloWorldFunction --stack-name sam-app
```

3. 的 Amazon SAM CLI 在云端调用你的函数并返回响应。下面是一个示例输出：

```
$ sam remote invoke HelloWorldFunction --stack-name sam-app

Invoking Lambda Function HelloWorldFunction
START RequestId: d5ef494b-5f45-4086-86fd-d7322fa1a1f9 Version: $LATEST
END RequestId: d5ef494b-5f45-4086-86fd-d7322fa1a1f9
REPORT RequestId: d5ef494b-5f45-4086-86fd-d7322fa1a1f9  Duration: 6.62 ms
 Billed Duration: 7 ms     Memory Size: 128 MB     Max Memory Used: 67 MB  Init
 Duration: 164.06 ms
{"statusCode":200,"body":"{\"message\":\"hello world\"}"}%
```

步骤 6：修改您的应用程序并将其同步到 Amazon Web Services 云

在此步骤中，您将使用 Amazon SAM CLI sam sync --watch命令将本地更改同步到 Amazon Web Services 云。

使用 sam 同步

1. 从 `sam-app` 项目目录，在命令行中运行以下命令：

```
$ sam sync --watch
```

2. 的 Amazon SAM CLI 提示您确认正在同步开发堆栈。由于该sam sync --watch命令会自动将本地更改实时部署到 Amazon Web Services 云 中，因此我们建议仅将其用于开发环境。

   的 Amazon SAM CLI 在开始监视本地更改之前执行初始部署。下面是一个示例输出：

```
$ sam sync --watch
The SAM CLI will use the AWS Lambda, Amazon API Gateway, and AWS StepFunctions APIs
 to upload your code without
performing a CloudFormation deployment. This will cause drift in your
 CloudFormation stack.
```

```
**The sync command should only be used against a development stack**.

Confirm that you are synchronizing a development stack.

Enter Y to proceed with the command, or enter N to cancel:
 [Y/n]: y
Queued infra sync. Waiting for in progress code syncs to complete...
Starting infra sync.
Manifest is not changed for (HelloWorldFunction), running incremental build
Building codeuri: /Users/.../Demo/sam-tutorial1/sam-app/hello_world runtime:
 python3.9 metadata: {} architecture: x86_64 functions: HelloWorldFunction
Running PythonPipBuilder:CopySource

Build Succeeded

Successfully packaged artifacts and wrote output template to file /var/
folders/45/5ct135bx3fn2551_ptl5g6_80000gr/T/tmpq3x9vh63.
Execute the following command to deploy the packaged template
sam deploy --template-file /var/folders/45/5ct135bx3fn2551_ptl5g6_80000gr/T/
tmpq3x9vh63 --stack-name <YOUR STACK NAME>


    Deploying with following values
    ===============================
    Stack name                   : sam-app
    Region                       : us-west-2
    Disable rollback             : False
    Deployment s3 bucket         : aws-sam-cli-managed-default-samclisam-s3-demo-
bucket-1a4x26zbcdkqr
    Capabilities                 : ["CAPABILITY_NAMED_IAM",
 "CAPABILITY_AUTO_EXPAND"]
    Parameter overrides          : {}
    Signing Profiles             : null

Initiating deployment
=====================


2023-03-15 13:10:05 - Waiting for stack create/update to complete

CloudFormation events from stack operations (refresh every 0.5 seconds)
---------------------------------------------------------------------------------------------
ResourceStatus                      ResourceType
 LogicalResourceId                   ResourceStatusReason
```

```
-----------------------------------------------------------------------------------------
UPDATE_IN_PROGRESS                  AWS::CloudFormation::Stack          sam-app
                         Transformation succeeded
CREATE_IN_PROGRESS                  AWS::CloudFormation::Stack
 AwsSamAutoDependencyLayerNestedSt   -
                                                                         ack
CREATE_IN_PROGRESS                  AWS::CloudFormation::Stack
 AwsSamAutoDependencyLayerNestedSt   Resource creation Initiated
                                                                         ack
CREATE_COMPLETE                     AWS::CloudFormation::Stack
 AwsSamAutoDependencyLayerNestedSt   -
                                                                         ack
UPDATE_IN_PROGRESS                  AWS::Lambda::Function
 HelloWorldFunction                  -
UPDATE_COMPLETE                     AWS::Lambda::Function
 HelloWorldFunction                  -
UPDATE_COMPLETE_CLEANUP_IN_PROGRE   AWS::CloudFormation::Stack          sam-app
                         -
SS
UPDATE_COMPLETE                     AWS::CloudFormation::Stack          sam-app
                         -
-----------------------------------------------------------------------------------------

CloudFormation outputs from deployed stack
-----------------------------------------------------------------------------------------
Outputs
-----------------------------------------------------------------------------------------
Key                 HelloWorldFunctionIamRole
Description         Implicit IAM Role created for Hello World function
Value               arn:aws:iam::012345678910:role/sam-app-
HelloWorldFunctionRole-15GLOUR9LMT1W

Key                 HelloWorldApi
Description         API Gateway endpoint URL for Prod stage for Hello World
 function
Value               https://ets1gv8lxi.execute-api.us-west-2.amazonaws.com/Prod/
hello/

Key                 HelloWorldFunction
Description         Hello World Lambda Function ARN
Value               arn:aws:lambda:us-west-2:012345678910:function:sam-app-
HelloWorldFunction-yQDNe17r9maD
-----------------------------------------------------------------------------------------
```

```
Stack update succeeded. Sync infra completed.

Infra sync completed.
CodeTrigger not created as CodeUri or DefinitionUri is missing for
 ServerlessRestApi.
```

接下来，您将修改 Lambda 函数代码。的 Amazon SAM CLI 将自动检测到此更改并将您的应用程序同步到 Amazon Web Services 云。

修改和同步应用程序

1. 在选定 IDE 中打开 `sam-app/hello_world/app.py` 文件。
2. 更改 `message` 然后保存文件。以下是示例：

```
import json
...
def lambda_handler(event, context):
    ...
    return {
        "statusCode": 200,
        "body": json.dumps({
            "message": "hello everyone!",
            ...
        }),
    }
```

3. 的 Amazon SAM CLI 检测到您的更改并将您的应用程序同步到。 Amazon Web Services 云下面是一个示例输出：

```
Syncing Lambda Function HelloWorldFunction...
Manifest is not changed for (HelloWorldFunction), running incremental build
Building codeuri: /Users/.../Demo/sam-tutorial1/sam-app/hello_world runtime:
 python3.9 metadata: {} architecture: x86_64 functions: HelloWorldFunction
Running PythonPipBuilder:CopySource
Finished syncing Lambda Function HelloWorldFunction.
```

4. 要验证您的更改，请再次向 API 端点发送 GET 请求。

```
$ curl https://ets1gv8lxi.execute-api.us-west-2.amazonaws.com/Prod/hello/
```

```
{"message": "hello everyone!"}
```

第 7 步：（可选）在本地测试应用程序

> **Note**
>
> 此为可选步骤。

> **Important**
>
> 此步骤需要 Docker 在你的本地计算机上。您必须具有…… Docker 已安装并配置为使用 Amazon SAM CLI 用于本地测试。有关更多信息，请参阅 安装 Docker。

在此步骤中，您将使用 Amazon SAM CLI sam local命令在本地测试您的应用程序。为了实现这一目标， Amazon SAM CLI 使用创建本地环境 Docker。 此本地环境模拟 Lambda 函数的基于云的执行环境。

您将执行以下操作：

1. 为 Lambda 函数创建本地环境并调用它。
2. 在本地托管您的 HTTP API 端点，并使用它调用 Lambda 函数。

要在本地调用 Lambda 函数

1. 从 `sam-app` 项目目录，在命令行中运行以下命令：

```
$ sam local invoke
```

2. 的 Amazon SAM CLI 创建本地 Docker 容器并调用你的函数。下面是一个示例输出：

```
$ sam local invoke
Invoking app.lambda_handler (python3.9)
Local image was not found.
Removing rapid images for repo public.ecr.aws/sam/emulation-python3.9
Building image.....................
Using local image: public.ecr.aws/lambda/python:3.9-rapid-x86_64.
```

```
Mounting /Users/.../Demo/sam-tutorial1/sam-app/.aws-sam/build/HelloWorldFunction
 as /var/task:ro,delegated inside runtime container
START RequestId: b046db01-2a00-415d-af97-35f3a02e9eb6 Version: $LATEST
END RequestId: b046db01-2a00-415d-af97-35f3a02e9eb6
REPORT RequestId: b046db01-2a00-415d-af97-35f3a02e9eb6    Init Duration: 1.01 ms
    Duration: 633.45 ms    Billed Duration: 634 ms    Memory Size: 128 MB    Max
 Memory Used: 128 MB
{"statusCode": 200, "body": "{\"message\": \"hello world\"}"}
```

在本地托管 API

1. 从 `sam-app` 项目目录，在命令行中运行以下命令：

```
$ sam local start-api
```

2. 的 Amazon SAM CLI 创建本地 Docker 为您的 Lambda 函数创建容器，并创建本地 HTTP 服务器来模拟您的 API 终端节点。下面是一个示例输出：

```
$ sam local start-api
Initializing the lambda functions containers.
Local image is up-to-date
Using local image: public.ecr.aws/lambda/python:3.9-rapid-x86_64.

Mounting /Users/.../Demo/sam-tutorial1/sam-app/.aws-sam/build/HelloWorldFunction
 as /var/task:ro,delegated inside runtime container
Containers Initialization is done.
Mounting HelloWorldFunction at http://127.0.0.1:3000/hello [GET]
You can now browse to the above endpoints to invoke your functions. You do not
 need to restart/reload SAM CLI while working on your functions, changes will be
 reflected instantly/automatically. If you used sam build before running local
 commands, you will need to re-run sam build for the changes to be picked up. You
 only need to restart SAM CLI if you update your AWS SAM template
2023-03-15 14:25:21 WARNING: This is a development server. Do not use it in a
 production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:3000
2023-03-15 14:25:21 Press CTRL+C to quit
```

3. 使用浏览器或命令行，向本地 API 端点发送 GET 请求。以下是使用 curl 命令的示例。

```
$ curl http://127.0.0.1:3000/hello
```

```
{"message": "hello world"}
```

第 8 步：从中删除您的应用程序 Amazon Web Services 云

在此步骤中，您将使用 Amazon SAM CLI sam delete命令从。中删除您的应用程序 Amazon Web Services 云。

要从中删除您的应用程序 Amazon Web Services 云

1. 从 `sam-app` 项目目录，在命令行中运行以下命令：

```
$ sam delete
```

2. 的 Amazon SAM CLI 会要求你确认。然后，它将删除您的应用程序的 Amazon S3 存储桶和 Amazon CloudFormation 堆栈。下面是一个示例输出：

```
$ sam delete
    Are you sure you want to delete the stack sam-app in the region us-west-2 ? [y/
N]: y
    Are you sure you want to delete the folder sam-app in S3 which contains the
 artifacts? [y/N]: y
    - Deleting S3 object with key c6ce8fa8b5a97dd022ecd006536eb5a4
    - Deleting S3 object with key 5d513a459d062d644f3b7dd0c8b56a2a.template
    - Deleting S3 object with key sam-app/2bebf67c79f6a743cc5312f6dfc1efee.template
    - Deleting S3 object with key sam-app/6b208d0e42ad15d1cee77d967834784b.template
    - Deleting S3 object with key sam-app/da3c598813f1c2151579b73ad788cac8
    - Deleting S3 object with key sam-app/f798cdd93cee188a71d120f14a035b11
    - Deleting Cloudformation stack sam-app

Deleted successfully
```

故障排除

要排除故障 Amazon SAM CLI，请参阅 Amazon SAM CLI 故障排除。

了解更多

要继续了解 Amazon SAM，请参阅以下资源：

- 完整 Amazon SAM 研讨会 - 旨在指导您使用 Amazon SAM 提供的许多主要功能的研讨会。

- 与 SAM 的会话 — 由我们的 Amazon 无服务器开发者倡导者团队制作的关于使用的 Amazon SAM视频系列。
- Serverless Land – 汇集了关于 Amazon 无服务器项目的最新信息、博客、视频、代码和学习资源的网站。

如何使用 Amazon Serverless Application Model (Amazon SAM)

用于开发应用程序的主要工具是以及Amazon SAM 模板Amazon SAMCLI和 Amazon SAM 项目（这是您的应用程序项目目录）。您可以使用以下工具来：

1. 开发应用程序（其中包括初始化应用程序、定义资源和构建应用程序）。
2. 测试您的应用程序.
3. 调试应用程序.
4. 部署应用程序和资源.
5. 监控应用程序.

Amazon SAM 在您运行sam init命令并完成其后续工作流程后创建您的 Amazon SAM 项目。您可以通过向 Amazon SAM 项目添加代码来定义您的无服务器应用程序。虽然您的 Amazon SAM 项目由一组文件和文件夹组成，但其中最重要的文件是您的 Amazon SAM 模板（已命名`template.yaml`）。在此模板中，您可以编写代码来表达资源、事件源映射以及定义无服务器应用程序的其他属性。

Amazon SAMCLI包含您在 Amazon SAM 项目中使用的命令存储库。更具体地说， Amazon SAMCLI 就是用来构建、转换、部署、调试、打包、初始化和同步 Amazon SAM 项目的工具。换句话说，它是你用来将 Amazon SAM 项目变成无服务器应用程序的工具。

主题

- 的 Amazon SAM CLI
- Amazon SAM 项目和 Amazon SAM 模板

的 Amazon SAM CLI

Amazon Serverless Application Model 命令行界面 (Amazon SAM CLI) 是您用来在 Amazon SAM 应用程序项目目录上运行命令并最终将其转换为无服务器应用程序的工具。更具体地说， Amazon SAM CLI 允许您构建、转换、部署、调试、打包、初始化和同步 Amazon SAM 应用程序项目目录。

的 Amazon SAM CLI Amazon SAM 模板附带支持的第三方集成，用于构建和运行您的无服务器应用程序。

主题

- 怎么样 Amazon SAM CLI 命令已记录在案
- 正在配置 Amazon SAM CLI
- Amazon SAM CLI 核心命令

怎么样 Amazon SAM CLI 命令已记录在案

Amazon SAM CLI 使用以下格式对　命令进行记录：

- 提示 — Linux prompt 默认记录在案，显示为 ($)。对于以下命令 Windows 具体的，(>) 用作提示。请勿在键入命令时包含提示符。
- 目录 – 当必须从特定目录执行命令时，目录名称将显示在提示符符号之前。
- 用户输入 – 您在命令行处输入的命令文本采用 **`user input`** 格式。
- 可替换文本-可变文本（例如文件名和参数）的格式为*`replaceable text`*。在多行命令中或需要特定键盘输入的命令中，键盘输入也可显示为可替换文本。例如，*`ENTER`*。
- 输出 – 作为对命令的响应而返回的输出格式为 `computer output`。

下面是 `sam deploy` 命令和输出的示例：

```
$ sam deploy --guided --template template.yaml

Configuring SAM deploy
======================

    Looking for config file [samconfig.toml] :  Found
    Reading default arguments  :  Success

    Setting default arguments for 'sam deploy'
    =========================================
    Stack Name [sam-app]: ENTER
    AWS Region [us-west-2]: ENTER
    #Shows you resources changes to be deployed and require a 'Y' to initiate deploy
    Confirm changes before deploy [y/N]: ENTER
    #SAM needs permission to be able to create roles to connect to the resources in
 your template
    Allow SAM CLI IAM role creation [Y/n]: ENTER
    #Preserves the state of previously provisioned resources when an operation fails
    Disable rollback [y/N]: ENTER
    HelloWorldFunction may not have authorization defined, Is this okay? [y/N]: y
    Save arguments to configuration file [Y/n]: ENTER
```

```
    SAM configuration file [samconfig.toml]: ENTER
    SAM configuration environment [default]: ENTER
```

1. `sam deploy --guided --template template.yaml` 是您在命令行中输入的命令。
2. **`sam deploy --guided --template`** 应按原样提供。
3. *`template.yaml`*可以用您的特定文件名替换。
4. 输出开始于 `Configuring SAM deploy`。
5. 在输出中，*`ENTER`*并*`y`*指明您提供的可替换值。

正在配置 Amazon SAM CLI

的好处之一 Amazon SAM 是，它通过删除重复的任务来优化开发人员的时间。 Amazon SAM CLI 包括为此目的`samconfig`而命名的配置文件。默认情况下，没有配置 Amazon SAM CLI 是必需的，但您可以更新配置文件，允许改为在配置文件中引用自定义参数， Amazon SAM 从而使用较少的参数运行命令。下表中的示例显示了如何优化命令：

原始	使用 **`samconfig`** 进行优化
sam build --cached --parallel --use-containers	sam build
sam local invoke --env-vars locals.json	sam local invoke
sam local start-api --env-vars locals.json --warm-containers EAGER	sam local start-api

的 Amazon SAM CLI 提供了一组命令来帮助开发人员创建、开发和部署无服务器应用程序。其中每个命令都可以根据应用程序和开发人员的偏好使用可选标志进行配置。有关更多信息，请参阅 Amazon SAM。 CLI 内容在 GitHub

本节中的主题介绍如何创建 Amazon SAM CLI 配置文件 和自定义默认设置，以优化无服务器应用程序的开发时间。

主题

- 如何创建配置文件（samconfig 文件）
- 配置项目设置
- 配置凭证和基本设置

如何创建配置文件（samconfig 文件）

的 Amazon SAM CLI 配置文件（文件名samconfig）是一个文本文件，通常使用 TOML 结构，但也可以采用 YAML。使用 Amazon 快速入门模板时，此文件是在您运行sam init命令时创建的。使用 sam deploy -\-guided 命令部署应用程序时，可以更新此文件。

部署完成后，如果使用默认值，则 samconfig 文件将包含一个名为 default 的配置文件。重新运行该deploy命令时，会 Amazon SAM 应用此配置文件中存储的配置设置。

该samconfig文件的好处是，除了 deploy 命令之外，还可以 Amazon SAM 存储任何其他可用命令的配置设置。除了在新部署时创建的这些值外，您还可以在samconfig文件中设置许多属性，这些属性可以简化开发人员工作流程的其他方面 Amazon SAM CLI.

配置项目设置

您可以指定特定于项目的设置，例如 Amazon SAM CLI 命令参数值，在配置文件中用于 Amazon SAM CLI。 有关此配置文件的更多信息，请参阅Amazon SAM CLI 配置文件。

使用配置文件

配置文件由环境、命令和参数值构成。有关更多信息，请参阅 配置文件基础。

配置新环境

1. 在配置文件中指定新环境。

   在以下示例中，指定了新的 prod 环境：

   TOML

   ```
   [prod.global.parameters]
   ```

   YAML

   ```
   prod:
     global:
       parameters:
   ```

2. 在配置文件的参数部分将参数值指定为键值对。

   在以下示例中，为 prod 环境指定了应用程序的堆栈名称。

TOML

```
[prod.global.parameters]
stack_name = "prod-app"
```

YAML

```
prod:
  global:
    parameters:
      stack_name: prod-app
```

3. 使用 `--config-env` 选项指定要使用的环境。

   以下是示例：

```
$ sam deploy --config-env "prod"
```

配置参数值

1. 指定 Amazon SAM CLI 你要为其配置参数值的命令。为所有人配置参数值 Amazon SAM CLI 命令，使用标`global`识符。

   在以下示例中，为 `default` 环境的 `sam deploy` 命令指定了参数值：

   TOML

```
[default.deploy.parameters]
confirm_changeset = true
```

   YAML

```
default:
  deploy:
    parameters:
      confirm_changeset: true
```

   以下是为所有参数指定参数值的示例 Amazon SAM CLI `default`环境中的命令：

TOML

```
[default.global.parameters]
stack_name = "sam-app"
```

YAML

```
default:
  global:
    parameters:
      stack_name: sam-app
```

2. 您也可以通过指定参数值并修改配置文件 Amazon SAM CLI 交互流程。

   以下是 `sam deploy --guided` 交互式流程的示例：

```
$ sam deploy --guided

Configuring SAM deploy
======================

    Looking for config file [samconfig.toml] :  Found
    Reading default arguments  :  Success

    Setting default arguments for 'sam deploy'
    =========================================
    Stack Name [sam-app]: ENTER
    AWS Region [us-west-2]: ENTER
    #Shows you resources changes to be deployed and require a 'Y' to initiate
 deploy
    Confirm changes before deploy [Y/n]: n
    #SAM needs permission to be able to create roles to connect to the resources in
 your template
    Allow SAM CLI IAM role creation [Y/n]: ENTER
    #Preserves the state of previously provisioned resources when an operation
 fails
    Disable rollback [y/N]: ENTER
    HelloWorldFunction may not have authorization defined, Is this okay? [y/N]: y
    Save arguments to configuration file [Y/n]: ENTER
    SAM configuration file [samconfig.toml]: ENTER
    SAM configuration environment [default]: ENTER
```

有关更多信息，请参阅 创建和修改配置文件。

示例

基本 TOML 示例

下面是samconfig.toml配置文件的示例：

```
...
version = 0.1

[default]
[default.global]
[default.global.parameters]
stack_name = "sam-app"

[default.build.parameters]
cached = true
parallel = true

[default.deploy.parameters]
capabilities = "CAPABILITY_IAM"
confirm_changeset = true
resolve_s3 = true

[default.sync.parameters]
watch = true

[default.local_start_api.parameters]
warm_containers = "EAGER"

[prod]
[prod.sync]
[prod.sync.parameters]
watch = false
```

基本 YAML 示例

下面是samconfig.yaml配置文件的示例：

```
version 0.1
default:
  global:
```

```
    parameters:
      stack_name: sam-app
  build:
    parameters:
      cached: true
      parallel: true
  deploy:
    parameters:
      capabilities: CAPABILITY_IAM
      confirm_changeset: true
      resolve_s3: true
  sync:
    parameters:
      watch: true
  local_start_api:
    parameters:
      warm_containers: EAGER
prod:
  sync:
    parameters:
      watch: false
```

配置凭证和基本设置

使用 Amazon Command Line Interface (Amazon CLI) 配置基本设置，例如 Amazon 凭据、默认区域名称和默认输出格式。配置完成后，您可以将这些设置与 Amazon SAM CLI。 要了解更多信息，请参阅《Amazon Command Line Interface 用户指南》中的以下内容：

- 配置基础知识
- 配置和凭证文件设置
- 的命名个人资料 Amazon CLI
- 使用启用了 IAM Identity Center 的命名配置文件

有关快速设置说明，请参阅步骤 5： Amazon CLI 使用配置 Amazon 凭证。

Amazon SAM CLI 核心命令

Amazon SAM CLI 有一些基本命令，可用于创建、构建、测试、部署和同步无服务器应用程序。下表列出了这些命令，并提供了每个命令的更多信息链接。

有关完整列表 Amazon SAM CLI 命令，请参阅Amazon SAM CLI 命令参考。

命令	作用	相关 主题
sam build	为开发人员工作流程中的后续步骤做好准备，例如本地测试或部署到 Amazon 云端。	• 建筑简介 Amazon SAM • sam build
sam deploy	使用 Amazon CloudFormation 将应用程序部署到 Amazon 云端。	• 使用部署简介 Amazon SAM • sam deploy
sam init	提供用于初始化和创建新的无服务器应用程序的选项。	• 在中创建您的应用程序 Amazon SAM • sam init
sam local	提供用于在本地测试无服务器应用程序的子命令。	• 使用测试简介 sam local 命令 • sam local generate-event • sam local invoke • sam local start-api • sam local start-lambda
sam remote invoke	提供一种与 Amazon 云中支持的 Amazon 资源进行交互的方式。	• 云端测试简介 sam remote invoke • sam remote invoke
sam remote test-event	提供一种访问和管理您的 Lambd Amazon a 函数的可共享测试事件的方法。	• 云测试简介 sam remote test-event • sam remote test-event
sam sync	提供将本地应用程序更改快速同步到 Amazon 云端的选项。	• 使用简介 sam sync 同步到 Amazon Web Services 云 • sam sync

Amazon SAM 项目和 Amazon SAM 模板

运行sam init命令并完成其后续工作流程后， Amazon SAM 创建您的应用程序项目目录，即您的 Amazon SAM 项目。您可以通过向 Amazon SAM 项目添加代码来定义您的无服务器应用程序。虽然您的 Amazon SAM 项目由一组文件和文件夹组成，但您主要使用的文件是您的 Amazon SAM 模板（已命名`template.yaml`）。在此模板中，您可以编写代码来表达资源、事件源映射以及定义无服务器应用程序的其他属性。

> **Note**
>
> 模板的一个关键元素是 Amazon SAM Amazon SAM 模板规范。本规范提供了简短的语法，与之相比 Amazon CloudFormation，它允许您使用更少的代码行来定义无服务器应用程序的资源、事件源映射 APIs、权限和其他属性。

本节详细介绍了如何使用 Amazon SAM 模板中的部分来定义资源类型、资源属性、数据类型、资源属性、内部函数和 API Gateway 扩展。

Amazon SAM 模板是 Amazon CloudFormation 模板的扩展，其独特的语法类型使用速记语法，代码行数少于。 Amazon CloudFormation在构建无服务器应用程序时，这可以加快开发速度。有关更多信息，请参阅Amazon SAM 资源和财产。有关 Amazon CloudFormation 模板的完整参考，请参阅Amazon CloudFormation 用户指南中的Amazon CloudFormation 模板参考。

在开发时，您通常会发现将应用程序代码分解为单独的文件是有益的，这样可以更好地组织和管理您的应用程序。这方面的一个基本例子是为 Amazon Lambda 函数代码使用单独的文件，而不是在 Amazon SAM 模板中包含此代码。为此，请将 Lambda 函数代码整理到项目的子目录中，然后在 () 模板中引用其本地路径。 Amazon Serverless Application Model Amazon SAM

主题

- Amazon SAM 模板解剖学
- Amazon SAM 资源和财产
- 生成的 Amazon CloudFormation 资源用于 Amazon SAM
- 支持的资源属性 Amazon SAM
- 适用于 API Gateway 的扩展 Amazon SAM
- 的内在函数 Amazon SAM

Amazon SAM 模板解剖学

Amazon SAM 模板文件严格遵循 Amazon CloudFormation 模板文件的格式，该格式在《Amazon CloudFormation 用户指南》的模板剖析中进行了描述。 Amazon SAM 模板文件和 Amazon CloudFormation 模板文件之间的主要区别如下：

- 转换声明。 Amazon SAM 模板文件需要声明 `Transform: AWS::Serverless-2016-10-31`。此声明将 Amazon CloudFormation 模板文件标识为 Amazon SAM 模板文件。有关转换的更多信息，请参阅《Amazon CloudFormation 用户指南》中的转换。
- 全局变量部分。该`Globals`部分是独一无二的 Amazon SAM。它定义了所有无服务器函数通用的属性，以及。 APIs所有 `AWS::Serverless::Function`、`AWS::Serverless::Api`、和 `AWS::Serverless::SimpleTable` 资源都继承 `Globals` 部分中定义的属性。有关该部分的更多信息，请参阅 Amazon SAM 模板的全局变量部分。
- 资源部分。在 Amazon SAM 模板中，该`Resources`部分可以包含 Amazon CloudFormation 资源和 Amazon SAM 资源的组合。有关 Amazon CloudFormation 资源的更多信息，请参阅Amazon CloudFormation 用户指南中的Amazon 资源和属性类型参考。有关 Amazon SAM 资源的更多信息，请参阅Amazon SAM 资源和财产。

Amazon SAM 模板文件的所有其他部分对应于同名的 Amazon CloudFormation 模板文件部分。

YAML

以下示例显示 YAML 格式的模板片段。

```
Transform: AWS::Serverless-2016-10-31

Globals:
  set of globals

Description:
  String

Metadata:
  template metadata

Parameters:
  set of parameters

Mappings:
  set of mappings
```

```
Conditions:
  set of conditions

Resources:
  set of resources

Outputs:
  set of outputs
```

模板部分

Amazon SAM 模板可以包括几个主要部分。仅 `Transform` 和 `Resources` 部分为必需。

您可以按任意顺序包含模板部分。但是，如果使用语言扩展，则应在无服务器转换之前（即 `AWS::Serverless-2016-10-31` 之前）添加 `AWS::LanguageExtensions`，如下例所示：

```
Transform:
  - AWS::LanguageExtensions
  - AWS::Serverless-2016-10-31
```

在您构建模板时，使用以下列表中显示的逻辑顺序可能会很有用。这是因为某一部分中的值可能引用前一部分中的值。

转换（必需）

对于 Amazon SAM 模板，必须包含此部分，其值为`AWS::Serverless-2016-10-31`。

其它变换是可选的。有关转换的更多信息，请参阅《Amazon CloudFormation 用户指南》中的转换。

全局变量（可选）

所有无服务器函数和简单表都通用的属性。 APIs所有 `AWS::Serverless::Function`、`AWS::Serverless::Api`、和 `AWS::Serverless::SimpleTable` 资源都继承 `Globals` 部分中定义的属性。

本节是独一无二的 Amazon SAM。 Amazon CloudFormation 模板中没有相应的部分。

Description（可选）

一个描述模板的文本字符串。

本节与 Amazon CloudFormation 模板`Description`部分直接对应。

元数据（可选）

提供有关模板的其他信息的对象。

本节与 Amazon CloudFormation 模板`Metadata`部分直接对应。

Parameters（可选）

要在运行时 (创建或更新堆栈时) 传递到模板的值。您可引用模板的 `Resources` 和 `Outputs` 部分中的参数。在 `Parameters` 部分中声明的对象会导致 sam deploy --guided 命令向用户显示附加提示。

使用`sam deploy`命令的`--parameter-overrides`参数和配置文件中的条目传入的值优先于模板文件中的条目。 Amazon SAM 有关 `sam deploy` 命令的更多信息，请参阅 Amazon SAM CLI 命令参考中的 sam deploy。有关配置文件的更多信息，请参阅Amazon SAM CLI 配置文件。

Mappings（可选）

可用来指定条件参数值的密钥和关键值的映射，与查找表类似。可以通过使用 `Resources` 和 `Outputs` 部分中的 `Fn::FindInMap` 内置函数将键与相应的值匹配。

本节与 Amazon CloudFormation 模板`Mappings`部分直接对应。

条件（可选）

用于控制是否创建某些资源或者是否在堆栈创建或更新过程中为某些资源属性分配值的条件。例如，您可以根据堆栈是用于生产环境还是用于测试环境来按照条件创建资源。

本节与 Amazon CloudFormation 模板`Conditions`部分直接对应。

Resources（必需）

堆栈资源及其属性，例如亚马逊弹性计算云 (Amazon EC2) 实例或亚马逊简单存储服务 (Amazon S3) Service 存储桶。您可引用模板的 `Resources` 和 `Outputs` 部分中的资源。

该部分与 Amazon CloudFormation 模板的 `Resources` 部分类似。在 Amazon SAM 模板中，除 Amazon SAM 资源外，此部分还可以包含 Amazon CloudFormation 资源。

Outputs（可选）

在您查看堆栈的属性时返回的值。例如，您可以为 S3 存储桶名称声明输出，然后调用 `aws cloudformation describe-stacks` Amazon Command Line Interface (Amazon CLI) 命令查看该名称。

该部分与 Amazon CloudFormation 模板的 `Outputs` 部分直接对应。

后续步骤

要下载和部署包含 Amazon SAM 模板文件的示例无服务器应用程序，请参阅入门 Amazon SAM并按照中的教程：使用以下命令部署 Hello World 应用程序 Amazon SAM说明进行操作。

Amazon SAM 模板的全局变量部分

有时，您在 Amazon SAM 模板中声明的资源具有通用配置。例如，应用程序可能包含多个具有相同的 `Runtime`、`Memory`、`VPCConfig`、`Environment` 和 `Cors` 配置的 `AWS::Serverless::Function` 资源。与其在每个资源中复制这些信息，不如在 `Globals` 部分中声明一次，然后让您的资源继承它们。

该`Globals`部分支持以下 Amazon SAM 资源类型：

- `AWS::Serverless::Api`
- `AWS::Serverless::Function`
- `AWS::Serverless::HttpApi`
- `AWS::Serverless::SimpleTable`
- `AWS::Serverless::StateMachine`

示例：

```
Globals:
  Function:
    Runtime: nodejs12.x
    Timeout: 180
    Handler: index.handler
    Environment:
      Variables:
        TABLE_NAME: data-table

Resources:
  HelloWorldFunction:
    Type: AWS::Serverless::Function
    Properties:
      Environment:
        Variables:
          MESSAGE: "Hello From SAM"

  ThumbnailFunction:
```

```
    Type: AWS::Serverless::Function
    Properties:
      Events:
        Thumbnail:
          Type: Api
          Properties:
            Path: /thumbnail
            Method: POST
```

在此示例中，`HelloWorldFunction` 和 `ThumbnailFunction` 都使用如下配置：`Runtime` 为“nodejs12.x”，`Timeout` 为“180”秒，`Handler` 为“index.handler”。除了继承的 TABLE_NAME 之外，`HelloWorldFunction` 还会添加 MESSAGE 环境变量。`ThumbnailFunction` 会继承所有 `Globals` 属性并添加 API 事件源。

支持的资源和属性

Amazon SAM 支持以下资源和属性。

```
Globals:
  Api:
    AccessLogSetting:
    Auth:
    BinaryMediaTypes:
    CacheClusterEnabled:
    CacheClusterSize:
    CanarySetting:
    Cors:
    DefinitionUri:
    Domain:
    EndpointConfiguration:
    GatewayResponses:
    MethodSettings:
    MinimumCompressionSize:
    Name:
    OpenApiVersion:
    PropagateTags:
    TracingEnabled:
    Variables:

  Function:
    Architectures:
    AssumeRolePolicyDocument:
    AutoPublishAlias:
```

```
    CodeSigningConfigArn:
    CodeUri:
    DeadLetterQueue:
    DeploymentPreference:
    Description:
    Environment:
    EphemeralStorage:
    EventInvokeConfig:
    FileSystemConfigs:
    FunctionUrlConfig:
    Handler:
    KmsKeyArn:
    Layers:
    LoggingConfig:
    MemorySize:
    PermissionsBoundary:
    PropagateTags:
    ProvisionedConcurrencyConfig:
    RecursiveLoop:
    ReservedConcurrentExecutions:
    RolePath:
    Runtime:
    RuntimeManagementConfig:
    SnapStart:
    SourceKMSKeyArn:
    Tags:
    Timeout:
    Tracing:
    VpcConfig:

  HttpApi:
    AccessLogSettings:
    Auth:
    PropagateTags:
    StageVariables:
    Tags:

  SimpleTable:
    SSESpecification:

  StateMachine:
    PropagateTags:
```

> **Note**
>
> 系统不支持任何未包含在前面列表中的资源和属性。不支持它们的一些原因包括：1) 它们会带来潜在的安全问题，或者 2) 它们使模板难以理解。

隐含的 APIs

Amazon SAM APIs当您在本`Events`节中声明 API 时会创建隐式的。您可以使用`Globals`覆盖隐式的所有属性 APIs。

可覆盖属性

资源可以覆盖您在 `Globals` 部分声明的属性。例如，您可以向环境变量映射中添加新变量，也可以覆盖全局声明的变量。但是资源无法删除 `Globals` 部分中指定的属性。

更笼统地说，`Globals` 部分会声明所有资源共享的属性。有些资源可以为全局声明的属性提供新的值，但它们无法将其删除。如果一些资源使用某属性而其他资源不使用，则不得在 `Globals` 部分中声明它们。

以下部分描述了覆盖如何应用于不同的数据类型。

替换原始数据类型

原始数据类型包括字符串、数字、布尔值等。

`Resources` 部分中指定的值替换 `Globals` 部分中的值。

示例：

```
Globals:
  Function:
    Runtime: nodejs12.x

Resources:
  MyFunction:
    Type: AWS::Serverless::Function
    Properties:
      Runtime: python3.9
```

`MyFunction` 的 `Runtime` 设置为 `python3.9`。

合并映射

映射也称为字典或键值对的集合。

Resources 部分中的映射条目与全局映射条目合并。如果存在重复项，则 Resource 部分条目将覆盖 Globals 部分条目。

示例：

```
Globals:
  Function:
    Environment:
      Variables:
        STAGE: Production
        TABLE_NAME: global-table

Resources:
  MyFunction:
    Type: AWS::Serverless::Function
    Properties:
      Environment:
        Variables:
          TABLE_NAME: resource-table
          NEW_VAR: hello
```

MyFunction 的环境变量设置如下：

```
{
  "STAGE": "Production",
  "TABLE_NAME": "resource-table",
  "NEW_VAR": "hello"
}
```

添加列表

列表也称为数组。

Globals 部分中的列表条目添加到 Resources 部分的列表之前。

示例：

```
Globals:
```

```
  Function:
    VpcConfig:
      SecurityGroupIds:
        - sg-123
        - sg-456

Resources:
  MyFunction:
    Type: AWS::Serverless::Function
    Properties:
      VpcConfig:
        SecurityGroupIds:
          - sg-first
```

针对 `MyFunction` 的 `VpcConfig` 的 `SecurityGroupIds` 设置如下：

```
[ "sg-123", "sg-456", "sg-first" ]
```

Amazon SAM 资源和财产

本节介绍特定于的资源和属性类型。 Amazon SAM您可以使用 Amazon SAM 速记语法定义这些资源和属性。 Amazon SAM 还支持 Amazon CloudFormation 资源和属性类型。有关所有 Amazon 资源和属性类型 Amazon CloudFormation 及 Amazon SAM 支持的参考信息，请参阅《Amazon CloudFormation 用户指南》中的Amazon 资源和属性类型参考。

主题

- AWS::Serverless::Api
- AWS::Serverless::Application
- AWS::Serverless::Connector
- AWS::Serverless::Function
- AWS::Serverless::GraphQLApi
- AWS::Serverless::HttpApi
- AWS::Serverless::LayerVersion
- AWS::Serverless::SimpleTable
- AWS::Serverless::StateMachine

AWS::Serverless::Api

创建一组可通过 HTTPS 端点调用的 Amazon API Gateway 资源和方法。

无需将AWS::Serverless::Api资源明确添加到 Amazon 无服务器应用程序定义模板中。这种类型的资源是通过 AWS::Serverless::Function 资源上定义的 Api 事件的并集隐式创建的，这些资源定义于未引用 AWS::Serverless::Api 资源的模板。

应使用AWS::Serverless::Api资源来定义和记录 API OpenApi，这样可以更好地配置底层 Amazon API Gateway 资源。

我们建议您使用 Amazon CloudFormation 挂钩或 IAM 策略来验证 API Gateway 资源是否附加了授权者来控制对它们的访问。

有关使用 Amazon CloudFormation 挂钩的更多信息，请参阅 Amazon CloudFormation CLI 用户指南和apigw-enforce-authorizer GitHub 存储库中的注册挂钩。

有关使用 IAM 策略的更多信息，请参阅《API Gateway 开发人员指南》中的要求 API 路由具有授权。

> **Note**
>
> 部署到时 Amazon CloudFormation， Amazon SAM 会将您的 Amazon SAM 资源转换为 Amazon CloudFormation 资源。有关更多信息，请参阅 生成的 Amazon CloudFormation 资源用于 Amazon SAM。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
Type: AWS::Serverless::Api
Properties:
  AccessLogSetting: AccessLogSetting
  AlwaysDeploy: Boolean
  ApiKeySourceType: String
  Auth: ApiAuth
  BinaryMediaTypes: List
  CacheClusterEnabled: Boolean
  CacheClusterSize: String
  CanarySetting: CanarySetting
  Cors: String | CorsConfiguration
```

```
  DefinitionBody: JSON
  DefinitionUri: String | ApiDefinition
  Description: String
  DisableExecuteApiEndpoint: Boolean
  Domain: DomainConfiguration
  EndpointConfiguration: EndpointConfiguration
  FailOnWarnings: Boolean
  GatewayResponses: Map
  MergeDefinitions: Boolean
  MethodSettings: MethodSettings
  MinimumCompressionSize: Integer
  Mode: String
  Models: Map
  Name: String
  OpenApiVersion: String
  PropagateTags: Boolean
  Policy: JSON
  StageName: String
  Tags: Map
  TracingEnabled: Boolean
  Variables: Map
```

属性

`AccessLogSetting`

配置某个阶段的访问日志设置。

类型：AccessLogSetting

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::ApiGateway::Stage`资源的`AccessLogSetting`属性。

`AlwaysDeploy`

即使未检测到 API 的更改，也要始终部署 API。

类型：布尔值

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`ApiKeySourceType`

用于根据使用计划对请求进行计量的 API 键的源。有效值为 `HEADER` 和 `AUTHORIZER`。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::ApiGateway::RestApi`资源的`ApiKeySourceType`属性。

`Auth`

配置授权，以控制对 API Gateway API 的访问。

有关使用配置访问权限的更多信息， Amazon SAM 请参阅使用您的 Amazon SAM 模板控制 API 访问权限。有关展示如何覆盖全局授权者的示例，请参见the section called “覆盖 Amazon API Gateway REST API 的全局授权方”。

类型：ApiAuth

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`BinaryMediaTypes`

您的 API 可能返回的 MIME 类型列表。使用它来启用对的二进制支持APIs。在 mime 类型中使用 ~1 而不是 /。

类型：列表

必需：否

Amazon CloudFormation 兼容性：此属性类似于`AWS::ApiGateway::RestApi`资源的`BinaryMediaTypes`属性。的列表 BinaryMediaTypes 已添加到 Amazon CloudFormation 资源和 OpenAPI 文档中。

`CacheClusterEnabled`

指示是否为阶段启用缓存。要缓存响应，还必须在 `MethodSettings` 下将 `CachingEnabled` 设置为 `true`。

类型：布尔值

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::ApiGateway::Stage`资源的`CacheClusterEnabled`属性。

`CacheClusterSize`

阶段的缓存群集大小。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::ApiGateway::Stage`资源的`CacheClusterSize`属性。

`CanarySetting`

将金丝雀设置配置为常规部署的某个阶段。

类型：CanarySetting

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::ApiGateway::Stage`资源的`CanarySetting`属性。

`Cors`

管理所有 API Gateway 的跨源资源共享 (CORS)。 APIs使用字符串指定允许使用的域，或使用其他 Cors 配置指定词典。

> **Note**
>
> CORS Amazon SAM 需要修改你的 OpenAPI 定义。在 `DefinitionBody` 中创建内联 OpenAPI 定义以开启 CORS。

有关 CORS 的更多信息，请参阅《API Gateway 开发人员指南》中的为 API Gateway REST API 资源启用 CORS。

类型：字符串 | CorsConfiguration

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`DefinitionBody`

描述您的 API 的 OpenAPI 规范。如果 `DefinitionUri` 和 `DefinitionBody` 均未指定，SAM 将根据模板配置为您生成 `DefinitionBody`。

要引用本地人 OpenAPI 定义你的 API 的文件，请使用`AWS::Include`转换。要了解更多信息，请参阅如何 Amazon SAM 上传本地文件。

类型：JSON

必需：否

Amazon CloudFormation 兼容性：此属性类似于`AWS::ApiGateway::RestApi`资源的Body属性。如果提供了某些属性，则可以在将内容传递给 DefinitionBody 之前在中插入或修改内容 CloudFormation。属性包括 `Auth`、`BinaryMediaTypes`、`Cors`、`GatewayResponses`、`Models`、和 `AWS::Serverless::Function` 相应 Api 类型的 `EventSource`。

`DefinitionUri`

Amazon S3 Uri、本地文件路径或定义 API 的 OpenAPI 文档的位置对象。此属性引用的 Amazon S3 对象必须是有效的 OpenAPI 文件。如果 `DefinitionUri` 和 `DefinitionBody` 均未指定，SAM 将根据模板配置为您生成 `DefinitionBody`。

如果提供了本地文件路径，则模板必须经过包含 `sam deploy` 或 `sam package` 命令的工作流程，才能使定义正确转换。

引用的外部 OpenApi 文件不支持内部函数。`DefinitionUri`改用带有 Include 转换的`DefinitionBody`属性将 OpenApi 定义导入到模板中。

类型：字符串 | ApiDefinition

必需：否

Amazon CloudFormation 兼容性：此属性类似于`AWS::ApiGateway::RestApi`资源的`BodyS3Location`属性。嵌套的 Amazon S3 属性的命名有所不同。

`Description`

Api 资源的描述。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::ApiGateway::RestApi`资源的`Description`属性。

`DisableExecuteApiEndpoint`

指定客户端是否可以使用默认 `execute-api` 端点调用您的 API。默认情况下，客户端可以使用默认 `https://{api_id}.execute-api.{region}.amazonaws.com` 调用您的 API。如果要求客户端使用自定义域名来调用 API，请指定 `True`。

类型：布尔值

必需：否

Amazon CloudFormation 兼容性：此属性类似于`AWS::ApiGateway::RestApi`资源的 `DisableExecuteApiEndpoint`属性。它直接传递给 `x-amazon-apigateway-endpoint-configuration` 扩展的 `disableExecuteApiEndpoint` 属性，然后添加到 `AWS::ApiGateway::RestApi` 资源的 `Body` 属性中。

`Domain`

为此 API Gateway API 配置自定义域。

类型：DomainConfiguration

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`EndpointConfiguration`

REST API 的端点类型。

类型：EndpointConfiguration

必需：否

Amazon CloudFormation 兼容性：此属性类似于`AWS::ApiGateway::RestApi`资源的`EndpointConfiguration`属性。嵌套配置属性的命名有所不同。

`FailOnWarnings`

指定在遇到警告时是回滚 (`true`) 还是不回滚 (`false`) API 创建。默认值为 `false`。

类型：布尔值

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::ApiGateway::RestApi`资源的`FailOnWarnings`属性。

`GatewayResponses`

为 API 配置网关响应。网关响应是 API Gateway 直接返回或使用 Lambda 授权方返回的响应。有关更多信息，请参阅网关响应的 Api Gateway OpenApi 扩展文档。

类型：映射

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`MergeDefinitions`

Amazon SAM 生成一个 OpenAPI 来自你的 API 事件源的规范。指定`true`将其 Amazon SAM 合并到内联中 OpenAPI 在您的`AWS::Serverless::Api`资源中定义的规范。指定 `false` 以不合并。

`MergeDefinitions` 需要定义 `AWS::Serverless::Api` 的 `DefinitionBody` 属性。`MergeDefinitions` 与 `AWS::Serverless::Api` 的 `DefinitionUri` 属性不兼容。

默认值：`false`

类型：布尔值

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`MethodSettings`

配置 API 阶段的所有设置，包括日志、指标、CacheTTL、节流。

类型： MethodSetting 列表

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::ApiGateway::Stage`资源的`MethodSettings`属性。

`MinimumCompressionSize`

允许根据客户端的 Accept-Encoding 标头压缩响应正文。当响应正文大小大于或等于您配置的阈值时，就会触发压缩。最大正文大小阈值为 10 MB（10,485,760 字节）。- 支持以下压缩类型：gzip、deflate 和身份。

类型：整数

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::ApiGateway::RestApi`资源的`MinimumCompressionSize`属性。

`Mode`

此属性仅在您使用 OpenAPI 定义 REST API 时才适用。`Mode` 确定 API Gateway 如何处理资源更新。有关更多信息，请参阅的 Mode 属性 AWS::ApiGateway::RestApi 资源类型。

有效值：`overwrite` 或 `merge`

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::ApiGateway::RestApi`资源的`Mode`属性。

`Models`

您的 API 方法要使用的架构。可以使用 JSON 或 YAML 描述这些架构。有关示例模型，请参阅本页底部的“示例”部分。

类型：映射

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`Name`

API Gateway RestApi 资源的名称

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::ApiGateway::RestApi`资源的`Name`属性。

`OpenApiVersion`

OpenApi 要使用的版本。这可以是 `2.0` Swagger 规范，也可以是 OpenApi 3.0 版本之一，比如。`3.0.1`有关 OpenAPI 的更多信息，请参阅 OpenAPI 规范。

> **Note**
>
> Amazon SAM 创建一个默认情况下名为`Stage`的阶段。将此属性设置为任何有效值将阻止阶段 `Stage` 的创建。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`PropagateTags`

指明是否将 `Tags` 属性中的标签传递给 AWS::Serverless::Api 生成的资源。指定 `True` 以在生成的资源中传播标签。

类型：布尔值

必需：否

默认值：`False`

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`Policy`

包含 API 权限的策略文档。要设置策略的 ARN，请使用 `!Join` 内部函数（`""` 作为分隔符）以及值 `"execute-api:/"` 和 `"*"`。

类型：JSON

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::ApiGateway::RestApi`资源的 Policy 属性。

`StageName`

阶段的名称，API Gateway 将它用作调用的统一资源标识符 (URI) 中的第一个路径部分。

要引用阶段资源，请使用 `<api-logical-id>.Stage`。有关指定 AWS::Serverless::Api 资源时引用生成资源的更多信息，请参阅 Amazon CloudFormation 何时生成的资源 AWS::Serverless::Api 已指定。有关生成的 Amazon CloudFormation 资源的一般信息，请参阅生成的 Amazon CloudFormation 资源用于 Amazon SAM。

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性类似于`AWS::ApiGateway::Stage`资源的`StageName`属性。它在 SAM 中为必需，但在 API Gateway 中并非必需

附加说明：隐式 API 的阶段名称为 “Prod”。

`Tags`

指定要添加到此 API Gateway 阶段的标签的映射（字符串到字符串）。有关标签的有效键和值的详细信息，请参阅《Amazon CloudFormation 用户指南》中的资源标签。

类型：映射

必需：否

Amazon CloudFormation 兼容性：此属性类似于`AWS::ApiGateway::Stage`资源的`Tags`属性。SAM 中的 Tags 属性由 Key: Value CloudFormation 对组成；其中包含标签对象的列表。

`TracingEnabled`

指示是否为阶段启用通过 X-Ray 进行的主动跟踪。有关 X-Ray 的更多信息，请参阅 API Gateway 开发者指南中的 APIs 使用 X-Ray 跟踪用户向 REST 发出的请求。

类型：布尔值

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::ApiGateway::Stage`资源的`TracingEnabled`属性。

`Variables`

一个定义阶段变量的映射 (字符串到字符串)，其中变量名作为键，变量值作为值。变量名称只能包含字母数字字符。值必须匹配以下正则表达式：`[A-Za-z0-9._~:/?#&=,-]+`。

类型：映射

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::ApiGateway::Stage`资源的`Variables`属性。

返回值

Ref

当向 `Ref` 内置函数提供此资源的逻辑 ID 时，它将返回底层 API Gateway API 的 ID。

有关使用 `Ref` 函数的更多信息，请参阅《Amazon CloudFormation 用户指南》中的 `Ref`。

Fn:: GetAtt

`Fn::GetAtt` 返回一个此类型指定属性的值。以下为可用属性和示例返回值。

有关使用 `Fn::GetAtt` 的更多信息，请参阅《Amazon CloudFormation 用户指南》中的 `Fn::GetAtt`。

`RootResourceId`

`RestApi` 资源的根资源 ID，例如 `a0bc123d4e`。

示例

SimpleApiExample

一个 Hello World Amazon SAM 模板文件，其中包含带有 API 端点的 Lambda 函数。这是正在运行的无服务器应用程序的完整 Amazon SAM 模板文件。

YAML

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: Amazon SAM template with a simple API definition
```

```
Resources:
  ApiGatewayApi:
    Type: AWS::Serverless::Api
    Properties:
      StageName: prod
  ApiFunction: # Adds a GET method at the root resource via an Api event
    Type: AWS::Serverless::Function
    Properties:
      Events:
        ApiEvent:
          Type: Api
          Properties:
            Path: /
            Method: get
            RestApiId:
              Ref: ApiGatewayApi
      Runtime: python3.10
      Handler: index.handler
      InlineCode: |
        def handler(event, context):
            return {'body': 'Hello World!', 'statusCode': 200}
```

ApiCorsExample

一个 Amazon SAM 模板片段，其中包含在外部 Swagger 文件中定义的 API 以及 Lambda 集成和 CORS 配置。这只是显示AWS::Serverless::Api定义的 Amazon SAM 模板文件的一部分。

YAML

```
Resources:
  ApiGatewayApi:
    Type: AWS::Serverless::Api
    Properties:
      StageName: Prod
      # Allows www.example.com to call these APIs
      # SAM will automatically add AllowMethods with a list of methods for this API
      Cors: "'www.example.com'"
      DefinitionBody: # Pull in an OpenApi definition from S3
        'Fn::Transform':
          Name: 'AWS::Include'
          # Replace "bucket" with your bucket name
          Parameters:
            Location: s3://bucket/swagger.yaml
```

ApiCognitoAuthExample

包含使用 Amazon Cognito 授权针对该 API 的请求的 API 的 Amazon SAM 模板片段。这只是显示AWS::Serverless::Api定义的 Amazon SAM 模板文件的一部分。

YAML

```
Resources:
  ApiGatewayApi:
    Type: AWS::Serverless::Api
    Properties:
      StageName: Prod
      Cors: "'*'"
      Auth:
        DefaultAuthorizer: MyCognitoAuthorizer
        Authorizers:
          MyCognitoAuthorizer:
            UserPoolArn:
              Fn::GetAtt: [MyCognitoUserPool, Arn]
```

ApiModelsExample

带有包含模型架构的 API 的 Amazon SAM 模板片段。这只是 Amazon SAM 模板文件的一部分，显示了一个包含两个模型架构的AWS::Serverless::Api定义。

YAML

```
Resources:
  ApiGatewayApi:
    Type: AWS::Serverless::Api
    Properties:
      StageName: Prod
      Models:
        User:
          type: object
          required:
            - username
            - employee_id
          properties:
            username:
              type: string
            employee_id:
              type: integer
```

```
            department:
              type: string
        Item:
          type: object
          properties:
            count:
              type: integer
            category:
              type: string
            price:
              type: integer
```

缓存示例

一个 Hello World Amazon SAM 模板文件，其中包含带有 API 端点的 Lambda 函数。API 已为一种资源和方法启用缓存。有关缓存的更多信息，请参阅的《API Gateway 开发人员指南》中的启用 API 缓存以增强响应能力。

YAML

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: AWS SAM template with a simple API definition with caching turned on
Resources:
  ApiGatewayApi:
    Type: AWS::Serverless::Api
    Properties:
      StageName: prod
      CacheClusterEnabled: true
      CacheClusterSize: '0.5'
      MethodSettings:
        - ResourcePath: /
          HttpMethod: GET
          CachingEnabled: true
          CacheTtlInSeconds: 300
      Tags:
        CacheMethods: All

  ApiFunction: # Adds a GET method at the root resource via an Api event
    Type: AWS::Serverless::Function
    Properties:
      Events:
        ApiEvent:
```

```
          Type: Api
          Properties:
            Path: /
            Method: get
            RestApiId:
              Ref: ApiGatewayApi
      Runtime: python3.10
      Handler: index.handler
      InlineCode: |
        def handler(event, context):
            return {'body': 'Hello World!', 'statusCode': 200}
```

带有私有 API 的自定义域名示例

一个 Hello World Amazon SAM 模板文件，其中包含一个 Lambda 函数，该函数的 API 终端节点映射到私有域。该模板在 VPC 终端节点和私有域之间创建域访问关联。有关更多信息，请参阅 API Gateway APIs 中的私有自定义域名。

YAML

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: AWS SAM template configured with a custom domain using a private API

Parameters:
    DomainName:
      Type: String
      Default: mydomain.example.com
    CertificateArn:
      Type: String
    HostedZoneId:
      Type: String
    VpcEndpointId:
      Type: String
    VpcEndpointDomainName:
      Type: String
    VpcEndpointHostedZoneId:
      Type: String

Resources:
  MyFunction:
    Type: AWS::Serverless::Function
    Properties:
```

```
      InlineCode: |
        def handler(event, context):
            return {'body': 'Hello World!', 'statusCode': 200}
      Handler: index.handler
      Runtime: python3.13
      Events:
        Fetch:
          Type: Api
          Properties:
            RestApiId:
              Ref: MyApi
            Method: Get
            Path: /get
  MyApi:
    Type: AWS::Serverless::Api
    Properties:
      StageName: Prod
      EndpointConfiguration:
        Type: PRIVATE
        VPCEndpointIds:
        - !Ref VpcEndpointId
      Policy:
        Version: '2012-10-17'
        Statement:
        - Effect: Allow
          Principal: '*'
          Action: execute-api:Invoke
          Resource: execute-api:/*
        - Effect: Deny
          Principal: '*'
          Action: execute-api:Invoke
          Resource: execute-api:/*
          Condition:
            StringNotEquals:
              aws:SourceVpce: !Ref VpcEndpointId
      Domain:
        DomainName: !Ref DomainName
        CertificateArn: !Ref CertificateArn
        EndpointConfiguration: PRIVATE
        BasePath:
        - /
        Route53:
          HostedZoneId: !Ref HostedZoneId
          VpcEndpointDomainName: !Ref VpcEndpointDomainName
```

```
          VpcEndpointHostedZoneId: !Ref VpcEndpointHostedZoneId
        AccessAssociation:
          VpcEndpointId: !Ref VpcEndpointId
        Policy:
          Version: '2012-10-17'
          Statement:
          - Effect: Allow
            Principal: '*'
            Action: execute-api:Invoke
            Resource: execute-api:/*
          - Effect: Deny
            Principal: '*'
            Action: execute-api:Invoke
            Resource: execute-api:/*
            Condition:
              StringNotEquals:
                aws:SourceVpce: !Ref VpcEndpointId
```

ApiAuth

配置授权，以控制对 API Gateway API 的访问。

有关使用配置访问权限的更多信息和示例， Amazon SAM 请参阅使用您的 Amazon SAM 模板控制 API 访问权限。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
  AddApiKeyRequiredToCorsPreflight: Boolean
  AddDefaultAuthorizerToCorsPreflight: Boolean
  ApiKeyRequired: Boolean
  Authorizers: CognitoAuthorizer | LambdaTokenAuthorizer | LambdaRequestAuthorizer |
AWS_IAM
  DefaultAuthorizer: String
  InvokeRole: String
  ResourcePolicy: ResourcePolicyStatement
  UsagePlan: ApiUsagePlan
```

> **Note**
>
> 该Authorizers属性包括AWS_IAM，但不需要额外的配置AWS_IAM。有关示例，请参阅Amazon IAM。

属性

`AddApiKeyRequiredToCorsPreflight`

如果设置了 `ApiKeyRequired` 和 `Cors` 属性，则设置 `AddApiKeyRequiredToCorsPreflight` 会导致 API 密钥被添加到 `Options` 属性中。

类型：布尔值

必需：否

默认值：`True`

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`AddDefaultAuthorizerToCorsPreflight`

如果设置了 `DefaultAuthorizer` 和 `Cors` 属性，则设置 `AddDefaultAuthorizerToCorsPreflight` 会导致默认授权方被添加到 OpenAPI 部分的 `Options` 属性中。

类型：布尔值

必需：否

默认值：True

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`ApiKeyRequired`

如果设置为 true，则所有 API 事件都需要 API 密钥。有关 API 密钥的更多信息，请参阅《API Gateway 开发人员指南》中的使用 API 密钥创建和使用使用计划。

类型：布尔值

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`Authorizers`

用于控制对 API Gateway API 的访问的授权方。

有关更多信息，请参阅 使用您的 Amazon SAM 模板控制 API 访问权限。

类型:CognitoAuthorizer| LambdaTokenAuthorizer| LambdaRequestAuthorizer| AWS_IAM

必需：否

默认值：无

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

其他说明：SAM 在 Api 的 OpenApi 定义中添加了授权方。

`DefaultAuthorizer`

为 API Gateway API 指定默认授权方，默认情况下，该授权方将用于授权 API 调用。

> **Note**
> 如果将与此 API EventSource 关联的函数的 Api 配置为使用 IAM 权限，则必须将此属性设置为`AWS_IAM`，否则将导致错误。

类型：字符串

必需：否

默认值：无

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`InvokeRole`

将所有资源和方法的集成凭证设置为此值。

CALLER_CREDENTIALS 映射到 arn:aws:iam::*:<user>/*，后者使用调用者凭证来调用端点。

有效值: CALLER_CREDENTIALS, NONE, IAMRoleArn

类型：字符串

必需：否

默认值：CALLER_CREDENTIALS

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

ResourcePolicy

为 API 中的所有方法和路径配置资源策略。

类型：ResourcePolicyStatement

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

附加说明：也可以使用 ApiFunctionAuth 在各个 AWS::Serverless::Function 中定义此设置。这是 with 所必需 APIs 的EndpointConfiguration: PRIVATE。

UsagePlan

配置与此 API 关联的使用计划。有关使用计划的更多信息，请参阅《API Gateway 开发人员指南》中的使用 API 密钥创建和使用使用计划。

设置此 Amazon SAM 属性后，此属性会额外生成三个 Amazon CloudFormation 资源：AWS::ApiGateway::UsagePlan，一个 AWS::ApiGateway::UsagePlanKey，还有一个 AWS::ApiGateway::ApiKey。 有关此场景的信息，请参阅UsagePlan属性已指定。有关生成的 Amazon CloudFormation 资源的一般信息，请参阅生成的 Amazon CloudFormation 资源用于 Amazon SAM。

类型：ApiUsagePlan

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

示例

CognitoAuth

Cognito 身份验证示例

YAML

```
Auth:
  Authorizers:
    MyCognitoAuth:
     UserPoolArn:
       Fn::GetAtt:
         - MyUserPool
         - Arn
     AuthType: "COGNITO_USER_POOLS"
  DefaultAuthorizer: MyCognitoAuth
  InvokeRole: CALLER_CREDENTIALS
  AddDefaultAuthorizerToCorsPreflight: false
  ApiKeyRequired: false
  ResourcePolicy:
    CustomStatements: [{
      "Effect": "Allow",
      "Principal": "*",
      "Action": "execute-api:Invoke",
      "Resource": "execute-api:/Prod/GET/pets",
      "Condition": {
          "IpAddress": {
              "aws:SourceIp": "1.2.3.4"
          }
        }
    }]
    IpRangeDenylist:
      - "10.20.30.40"
```

Amazon IAM

Amazon IAM 示例

YAML

```
Auth:
  Authorizers: AWS_IAM
```

ApiUsagePlan

为 API Gateway API 配置使用计划。有关使用计划的更多信息，请参阅《API Gateway 开发人员指南》中的使用 API 密钥创建和使用使用计划。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
CreateUsagePlan: String
Description: String
Quota: QuotaSettings
Tags: List
Throttle: ThrottleSettings
UsagePlanName: String
```

属性

`CreateUsagePlan`

确定如何配置此使用计划。有效值包括 `PER_API`、`SHARED` 和 `NONE`。

`PER_API`创建 AWS::ApiGateway::UsagePlan, AWS::ApiGateway::ApiKey 和 AWS::ApiGateway::UsagePlanKey特定于此 API 的资源。这些资源的逻辑 IDs 分别为`<api-logical-id>``UsagePlan``<api-logical-id>``ApiKey``<api-logical-id>``UsagePlanKey`、和。

`SHARED`创建 AWS::ApiGateway::UsagePlan, AWS::ApiGateway::ApiKey 和 AWS::ApiGateway::UsagePlanKey在同一 Amazon SAM 模板中也包含的任何 API 之间共享`CreateUsagePlan: SHARED`的资源。这些资源的逻辑 IDs 分别为`ServerlessUsagePlan``ServerlessApiKey``ServerlessUsagePlanKey`、和。如果您使用此选项，我们建议您仅在一个 API 资源中为此使用计划添加其他配置，以避免定义冲突和状态不确定性。

`NONE` 禁止创建使用计划或将使用计划与此 API 关联。在 Amazon SAM 模板的全局变量部分 中指定了 `SHARED` 或 `PER_API` 的情况下才需要这样做。

有效值：`PER_API`、`SHARED` 和 `NONE`

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`Description`

使用计划的描述。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::ApiGateway::UsagePlan`资源的`Description`属性。

`Quota`

配置用户可在指定时间间隔内发出的请求的数量。

类型：QuotaSettings

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::ApiGateway::UsagePlan`资源的`Quota`属性。

`Tags`

与使用计划关联的任意标签（键值对）的数组。

此属性使用CloudFormation 标签类型。

类型：列表

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::ApiGateway::UsagePlan`资源的`Tags`属性。

`Throttle`

配置整体请求速率（每秒平均请求数）和突发容量。

类型：ThrottleSettings

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::ApiGateway::UsagePlan`资源的`Throttle`属性。

`UsagePlanName`

使用计划的名称。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::ApiGateway::UsagePlan`资源的`UsagePlanName`属性。

示例

UsagePlan

以下是使用计划示例。

YAML

```
Auth:
  UsagePlan:
    CreateUsagePlan: PER_API
    Description: Usage plan for this API
    Quota:
      Limit: 500
      Period: MONTH
    Throttle:
      BurstLimit: 100
      RateLimit: 50
    Tags:
      - Key: TagName
        Value: TagValue
```

CognitoAuthorizer

定义 Amazon Cognito 用户群体授权方。

有关更多信息以及示例，请参阅 使用您的 Amazon SAM 模板控制 API 访问权限。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
AuthorizationScopes: List
Identity: CognitoAuthorizationIdentity
UserPoolArn: String
```

属性

`AuthorizationScopes`

此授权方的授权范围列表。

类型：列表

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`Identity`

此属性可用于在授权方传入请求中指定 `IdentitySource`。

类型：CognitoAuthorizationIdentity

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`UserPoolArn`

可以引用用户群体/指定要向其添加此 cognito 授权方的用户群体 arn

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

示例

CognitoAuth

Cognito 身份验证示例

YAML

```
Auth:
  Authorizers:
    MyCognitoAuth:
      AuthorizationScopes:
        - scope1
        - scope2
      UserPoolArn:
        Fn::GetAtt:
          - MyCognitoUserPool
          - Arn
      Identity:
        Header: MyAuthorizationHeader
        ValidationExpression: myauthvalidationexpression
```

CognitoAuthorizationIdentity

此属性可用于在授权方的传入请求 IdentitySource 中指定。有关更多信息， IdentitySource 请参阅 A ApiGateway uthorizer OpenApi 扩展。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
  Header: String
  ReauthorizeEvery: Integer
  ValidationExpression: String
```

属性

`Header`

在 OpenApi 定义中为 “授权” 指定标题名称。

类型：字符串

必需：否

默认值：Authorization

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`ReauthorizeEvery`

time-to-live(TTL) 周期，以秒为单位，用于指定 API Gateway 缓存授权方结果的时长。如果您指定一个大于 0 的值，API Gateway 将缓存授权方响应。默认情况下，API Gateway 将此属性设为 300。最大值为 3600 秒（1 小时）。

类型：整数

必需：否

默认值：300

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`ValidationExpression`

指定验证传入身份的验证表达式

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

示例

CognitoAuthIdentity

YAML

```
Identity:
  Header: MyCustomAuthHeader
  ValidationExpression: Bearer.*
  ReauthorizeEvery: 30
```

LambdaRequestAuthorizer

配置 Lambda 授权方以通过 Lambda 函数控制对 API 的访问。

有关更多信息以及示例，请参阅 使用您的 Amazon SAM 模板控制 API 访问权限。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
DisableFunctionDefaultPermissions: Boolean
FunctionArn: String
FunctionInvokeRole: String
FunctionPayloadType: String
Identity: LambdaRequestAuthorizationIdentity
```

属性

`DisableFunctionDefaultPermissions`

指定`true`以 Amazon SAM 防止自动创建`AWS::Lambda::Permissions`资源以在您的`AWS::Serverless::Api`资源和授权者 Lambda 函数之间配置权限。

默认值：`false`

类型：布尔值

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`FunctionArn`

指定 Lambda 函数的函数 ARN，该函数为 API 提供授权。

> **Note**
>
> Amazon SAM 当为指定`AWS::Lambda::Permissions`资源时`FunctionArn`，将自动创建资源`AWS::Serverless::Api`。该 `AWS::Lambda::Permissions` 资源在您的 API 和授权方 Lambda 函数之间配置权限。

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`FunctionInvokeRole`

将授权者凭证添加到 Lambda 授权方的 OpenApi 定义中。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`FunctionPayloadType`

此属性可用于定义 API 的 Lambda 授权方的类型。

有效值：`TOKEN` 或 `REQUEST`

类型：字符串

必需：否

默认值：`TOKEN`

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`Identity`

此属性可用于在授权方传入请求中指定 `IdentitySource`。仅当属性 `FunctionPayloadType` 为 `REQUEST` 时，该属性是必需属性。

类型：LambdaRequestAuthorizationIdentity

必需：条件

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

示例

LambdaRequestAuth

YAML

```
Authorizers:
  MyLambdaRequestAuth:
    FunctionPayloadType: REQUEST
    FunctionArn:
      Fn::GetAtt:
        - MyAuthFunction
        - Arn
    FunctionInvokeRole:
      Fn::GetAtt:
        - LambdaAuthInvokeRole
        - Arn
    Identity:
      Headers:
        - Authorization1
```

LambdaRequestAuthorizationIdentity

此属性可用于在授权方的传入请求 IdentitySource 中指定。有关更多信息， IdentitySource 请参阅 A ApiGateway uthorizer OpenApi 扩展。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
  Context: List
  Headers: List
  QueryStrings: List
  ReauthorizeEvery: Integer
  StageVariables: List
```

属性

`Context`

将给定的上下文字符串转换为格式 `context.contextString` 的映射表达式。

类型：列表

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`Headers`

将标头转换为格式 `method.request.header.name` 的映射表达式的逗号分隔字符串。

类型：列表

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`QueryStrings`

将给定的查询字符串转换为格式 `method.request.querystring.queryString` 的映射表达式的逗号分隔字符串。

类型：列表

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`ReauthorizeEvery`

time-to-live(TTL) 周期，以秒为单位，用于指定 API Gateway 缓存授权方结果的时长。如果您指定一个大于 0 的值，API Gateway 将缓存授权方响应。默认情况下，API Gateway 将此属性设为 300。最大值为 3600 秒（1 小时）。

类型：整数

必需：否

默认值：300

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`StageVariables`

将给定的阶段变量转换为格式 `stageVariables.stageVariable` 的映射表达式的逗号分隔字符串。

类型：列表

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

示例

LambdaRequestIdentity

YAML

```
Identity:
  QueryStrings:
    - auth
  Headers:
    - Authorization
  StageVariables:
    - VARIABLE
  Context:
    - authcontext
  ReauthorizeEvery: 100
```

LambdaTokenAuthorizer

配置 Lambda 授权方以通过 Lambda 函数控制对 API 的访问。

有关更多信息以及示例，请参阅 使用您的 Amazon SAM 模板控制 API 访问权限。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
DisableFunctionDefaultPermissions: Boolean
```

```
FunctionArn: String
FunctionInvokeRole: String
FunctionPayloadType: String
Identity: LambdaTokenAuthorizationIdentity
```

属性

`DisableFunctionDefaultPermissions`

指定`true`以 Amazon SAM 防止自动创建`AWS::Lambda::Permissions`资源以在您的`AWS::Serverless::Api`资源和授权者 Lambda 函数之间配置权限。

默认值：`false`

类型：布尔值

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`FunctionArn`

指定 Lambda 函数的函数 ARN，该函数为 API 提供授权。

> **Note**
>
> Amazon SAM 当为指定`AWS::Lambda::Permissions`资源时`FunctionArn`，将自动创建资源`AWS::Serverless::Api`。该 `AWS::Lambda::Permissions` 资源在您的 API 和授权方 Lambda 函数之间配置权限。

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`FunctionInvokeRole`

将授权者凭证添加到 Lambda 授权方的 OpenApi 定义中。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`FunctionPayloadType`

此属性可用于定义 Api 的 Lambda 授权方的类型。

有效值：`TOKEN` 或 `REQUEST`

类型：字符串

必需：否

默认值：`TOKEN`

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`Identity`

此属性可用于在授权方传入请求中指定 `IdentitySource`。仅当属性 `FunctionPayloadType` 为 `REQUEST` 时，该属性是必需属性。

类型：LambdaTokenAuthorizationIdentity

必需：条件

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

示例

LambdaTokenAuth

YAML

```
Authorizers:
  MyLambdaTokenAuth:
```

```
    FunctionArn:
      Fn::GetAtt:
        - MyAuthFunction
        - Arn
    Identity:
      Header: MyCustomAuthHeader # OPTIONAL; Default: 'Authorization'
      ValidationExpression: mycustomauthexpression # OPTIONAL
      ReauthorizeEvery: 20 # OPTIONAL; Service Default: 300
```

BasicLambdaTokenAuth

YAML

```
Authorizers:
  MyLambdaTokenAuth:
    FunctionArn:
      Fn::GetAtt:
        - MyAuthFunction
        - Arn
```

LambdaTokenAuthorizationIdentity

此属性可用于在授权方的传入请求 IdentitySource 中指定。有关更多信息， IdentitySource 请参阅ApiGateway 授权器 OpenApi 扩展。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
  Header: String
  ReauthorizeEvery: Integer
  ValidationExpression: String
```

属性

`Header`

在 OpenApi 定义中为授权指定标题名称。

类型：字符串

必需：否

默认值：Authorization

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`ReauthorizeEvery`

time-to-live(TTL) 周期，以秒为单位，用于指定 API Gateway 缓存授权方结果的时长。如果您指定一个大于 0 的值，API Gateway 将缓存授权方响应。默认情况下，API Gateway 将此属性设为 300。最大值为 3600 秒（1 小时）。

类型：整数

必需：否

默认值：300

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`ValidationExpression`

指定验证传入身份的验证表达式。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

示例

LambdaTokenIdentity

YAML

```
Identity:
  Header: MyCustomAuthHeader
  ValidationExpression: Bearer.*
  ReauthorizeEvery: 30
```

ResourcePolicyStatement

为 API 的所有方法和路径配置资源策略。有关资源策略的更多信息，请参阅《API Gateway 开发人员指南》中的使用 API Gateway 资源策略控制 API 的访问权限。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
AwsAccountBlacklist: List
AwsAccountWhitelist: List
CustomStatements: List
IntrinsicVpcBlacklist: List
IntrinsicVpcWhitelist: List
IntrinsicVpceBlacklist: List
IntrinsicVpceWhitelist: List
IpRangeBlacklist: List
IpRangeWhitelist: List
SourceVpcBlacklist: List
SourceVpcWhitelist: List
```

属性

`AwsAccountBlacklist`

要封锁的 Amazon 账户。

类型：字符串列表

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`AwsAccountWhitelist`

要允许的 Amazon 账户。有关此属性的使用示例，请参阅本页底部的“示例”部分。

类型：字符串列表

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`CustomStatements`

适用于此 API 的自定义资源策略语句列表。有关此属性的使用示例，请参阅本页底部的“示例”部分。

类型：列表

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`IntrinsicVpcBlacklist`

要屏蔽的虚拟私有云列表 (VPCs)，其中每个 VPC 都指定为引用，例如动态引用或`Ref`内部函数。有关此属性的使用示例，请参阅本页底部的“示例”部分。

类型：列表

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`IntrinsicVpcWhitelist`

VPCs 要允许的列表，其中每个 VPC 都指定为引用，例如动态引用或`Ref`内部函数。

类型：列表

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`IntrinsicVpceBlacklist`

要阻止的 VPC 端点列表，其中每个 VPC 端点都指定为引用，例如动态引用或`Ref`内置函数。

类型：列表

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`IntrinsicVpceWhitelist`

要允许的 VPC 端点列表，其中每个 VPC 端点都指定为引用，例如动态引用或`Ref`内置函数。有关此属性的使用示例，请参阅本页底部的“示例”部分。

类型：列表

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`IpRangeBlacklist`

要阻止的 IP 地址或地址范围。有关此属性的使用示例，请参阅本页底部的“示例”部分。

类型：列表

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`IpRangeWhitelist`

要允许的 IP 地址或地址范围。

类型：列表

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`SourceVpcBlacklist`

要阻止的源 VPC 或 VPC 端点。源 VPC 名称必须以 `"vpc-"` 开头，源 VPC 端点名称必须以 `"vpce-"` 开头。有关此属性的使用示例，请参阅本页底部的“示例”部分。

类型：列表

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`SourceVpcWhitelist`

要允许的源 VPC 或 VPC 端点。源 VPC 名称必须以 `"vpc-"` 开头，源 VPC 端点名称必须以 `"vpce-"` 开头。

类型：列表

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

示例

资源策略示例

以下示例屏蔽两个 IP 地址和一个源 VPC，并允许一个 Amazon 账户。

YAML

```
Auth:
  ResourcePolicy:
    CustomStatements: [{
                        "Effect": "Allow",
                        "Principal": "*",
                        "Action": "execute-api:Invoke",
                        "Resource": "execute-api:/Prod/GET/pets",
                        "Condition": {
                          "IpAddress": {
                            "aws:SourceIp": "1.2.3.4"
                          }
                        }
                      }]
    IpRangeBlacklist:
      - "10.20.30.40"
      - "1.2.3.4"
    SourceVpcBlacklist:
      - "vpce-1a2b3c4d"
    AwsAccountWhitelist:
      - "111122223333"
```

```
    IntrinsicVpcBlacklist:
      - "{{resolve:ssm:SomeVPCReference:1}}"
      - !Ref MyVPC
    IntrinsicVpceWhitelist:
      - "{{resolve:ssm:SomeVPCEReference:1}}"
      - !Ref MyVPCE
```

ApiDefinition

定义 API 的 OpenAPI 文档。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
Bucket: String
Key: String
Version: String
```

属性

`Bucket`

存储了 OpenAPI 文件的 Amazon S3 桶的名称。

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::ApiGateway::RestApiS3Location`数据类型的`Bucket`属性。

`Key`

OpenAPI 文件的 Amazon S3 密钥。

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::ApiGateway::RestApiS3Location`数据类型的`Key`属性。

`Version`

对于受版本控制的对象，是 OpenAPI 文件的版本。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::ApiGateway::RestApiS3Location`数据类型的`Version`属性。

示例

定义 Uri 示例

API 定义示例

YAML

```
DefinitionUri:
  Bucket: amzn-s3-demo-bucket-name
  Key: mykey-name
  Version: 121212
```

CorsConfiguration

管理 API Gateway 的跨源资源共享 (CORS)。 APIs使用字符串指定允许使用的域，或使用其他 Cors 配置指定词典。

> **Note**
>
> CORS Amazon SAM 需要修改你的 OpenAPI 定义。在 `DefinitionBody` 中创建内联 OpenAPI 定义以开启 CORS。如果在 `CorsConfiguration` OpenAPI 定义中和属性级别设置了，则将其 Amazon SAM 合并。属性级别优先于 OpenAPI 定义。

有关 CORS 的更多信息，请参阅《API Gateway 开发人员指南》中的为 API Gateway REST API 资源启用 CORS。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```yaml
AllowCredentials: Boolean
AllowHeaders: String
AllowMethods: String
AllowOrigin: String
MaxAge: String
```

属性

`AllowCredentials`

表示是否允许请求包含凭证的布尔值。

类型：布尔值

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`AllowHeaders`

允许的标头字符串。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`AllowMethods`

包含允许的 HTTP 方法的字符串。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`AllowOrigin`

允许的源字符串。这可以是字符串格式的逗号分隔的列表。

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`MaxAge`

包含缓存 CORS 预检请求的秒数的字符串。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

示例

CorsConfiguration

CORS 配置示例。这只是 Amazon SAM 模板文件的一部分，其中显示了配置了 CORS 的AWS::Serverless::Api定义和. AWS::Serverless::Function 如果您使用 Lambda 代理集成或 HTTP 代理集成，则您的后端必须返回 `Access-Control-Allow-Origin`、`Access-Control-Allow-Methods` 和 `Access-Control-Allow-Headers` 标头。

YAML

```
Resources:
  ApiGatewayApi:
    Type: AWS::Serverless::Api
    Properties:
      StageName: Prod
      Cors:
        AllowMethods: "'POST, GET'"
        AllowHeaders: "'X-Forwarded-For'"
        AllowOrigin: "'https://example.com'"
        MaxAge: "'600'"
```

```
        AllowCredentials: true
  ApiFunction: # Adds a GET method at the root resource via an Api event
    Type: AWS::Serverless::Function
    Properties:
      Events:
        ApiEvent:
          Type: Api
          Properties:
            Path: /
            Method: get
            RestApiId:
              Ref: ApiGatewayApi
      Runtime: python3.10
      Handler: index.handler
      InlineCode: |
        import json
        def handler(event, context):
          return {
          'statusCode': 200,
          'headers': {
            'Access-Control-Allow-Headers': 'Content-Type',
            'Access-Control-Allow-Origin': 'https://example.com',
            'Access-Control-Allow-Methods': 'POST, GET'
            },
          'body': json.dumps('Hello from Lambda!')
          }
```

DomainConfiguration

为 API 配置自定义域。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
AccessAssociation: DomainAccessAssociation
BasePath: List
CertificateArn: String
DomainName: String
EndpointConfiguration: String
MutualTlsAuthentication: MutualTlsAuthentication
NormalizeBasePath: Boolean
```

```
  OwnershipVerificationCertificateArn: String
  Policy: Json
  Route53: Route53Configuration
  SecurityPolicy: String
```

属性

`AccessAssociation`

生成 `AWS::ApiGateway::DomainNameAccessAssociation`资源所需的配置。

Amazon SAM 生成一个 AWS::ApiGateway::DomainNameAccessAssociation设置此属性时的资源。有关生成的 Amazon CloudFormation 资源的信息，请参阅生成的 Amazon CloudFormation 资源用于 Amazon SAM。

类型：DomainAccessAssociation

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`BasePath`

要使用 Amazon API Gateway 域名配置的基本路径列表。

类型：列表

必需：否

默认值：/

Amazon CloudFormation 兼容性：此属性类似于`AWS::ApiGateway::BasePathMapping`资源的`BasePath`属性。 Amazon SAM 创建多个`AWS::ApiGateway::BasePathMapping`资源，每个资源在此属性中`BasePath`指定一个。

`CertificateArn`

Amazon 托管证书的亚马逊资源名称 (ARN) 此域名的终端节点。 Amazon Certificate Manager 是唯一支持的来源。

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性类似于`AWS::ApiGateway::DomainName`资源的`CertificateArn`属性。如果设置`EndpointConfiguration`为`REGIONAL`（默认值），则`CertificateArn`映射到 RegionalCertificateArnin `AWS::ApiGateway::DomainName`。如果设置`EndpointConfiguration`为`EDGE`，则`CertificateArn`映射到 CertificateArnin `AWS::ApiGateway::DomainName`。如果设置`EndpointConfiguration`为`PRIVATE`，则此属性将传递给 AWS::ApiGateway::DomainNameV2 资源。

其他说明：对于终`EDGE`端节点，您必须在`us-east-1` Amazon 区域中创建证书。

`DomainName`

API Gateway API 的自定义域名。不支持大写字母。

Amazon SAM 生成一个 AWS::ApiGateway::DomainName设置此属性时的资源。有关此场景的更多信息，请参阅DomainName属性已指定。有关生成的 Amazon CloudFormation 资源的信息，请参阅生成的 Amazon CloudFormation 资源用于 Amazon SAM。

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::ApiGateway::DomainName`资源的`DomainName`属性，或者在设置为`AWS::ApiGateway::DomainNameV2`时传递给 EndpointConfiguration 该属性`PRIVATE`。

`EndpointConfiguration`

定义要映射到自定义域的 API Gateway 端点的类型。此属性的值决定了该`CertificateArn`属性的映射方式 Amazon CloudFormation。

有效值：`EDGE`、`REGIONAL` 或 `PRIVATE`

类型：字符串

必需：否

默认值：`REGIONAL`

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`MutualTlsAuthentication`

自定义域名的相互传输层安全性协议（TLS）身份验证配置。

类型：MutualTlsAuthentication

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::ApiGateway::DomainName`资源的`MutualTlsAuthentication`属性。

`NormalizeBasePath`

表示 `BasePath` 属性定义的基本路径中是否允许非字母数字字符。如果设置为 `True`，则将从基本路径中移除非字母数字字符。

针对 `BasePath` 属性使用 `NormalizeBasePath`。

类型：布尔值

必需：否

默认值：True

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`OwnershipVerificationCertificateArn`

ACM 颁发的用于验证自定义域所有权的公有证书的 ARN。只有在配置双向 TLS 并且为 `CertificateArn` 指定了 ACM 导入的或私有 CA 证书 ARN 时才需要。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::ApiGateway::DomainName`资源的`OwnershipVerificationCertificateArn`属性。

`Policy`

要附加到 API Gateway 域名的 IAM 策略。仅在设置`EndpointConfiguration`为时适用`PRIVATE`。

类型：Json

必需：否

Amazon CloudFormation 兼容性：当设置为时`EndpointConfiguration`，此`Policy`属性将直接传递给`AWS::ApiGateway::DomainNameV2`资源的属性`PRIVATE`。有关有效政策文件的示例，请参阅 AWS::ApiGateway::DomainNameV2。

`Route53`

定义 Amazon Route 53 配置。

类型：Route53Configuration

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`SecurityPolicy`

此域名的 TLS 版本加密码套件。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::ApiGateway::DomainName`资源的`SecurityPolicy`属性，或者在设置为`AWS::ApiGateway::DomainNameV2`时传递给`EndpointConfiguration`该属性`PRIVATE`。对于`PRIVATE`端点，仅支持 TLS_1_2。

示例

DomainName

DomainName 示例

YAML

```
Domain:
  DomainName: www.example.com
  CertificateArn: arn-example
  EndpointConfiguration: EDGE
  Route53:
    HostedZoneId: Z1PA6795UKMFR9
  BasePath:
    - foo
```

```
    - bar
```

DomainAccessAssociation

在访问关联源和私有自定义域名之间配置域访问关联。唯一支持的访问关联源是 VPC 终端节点 ID。有关更多信息，请参阅 API Gateway APIs 中的私有自定义域名。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
VpcEndpointId: String
```

属性

`VpcEndpointId`

与 API Gateway VPC 服务关联的 VPC 接口终端节点的终端节点 ID。

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::ApiGateway::DomainNameAccessAssociation`资源的`AccessAssociationSource`属性。

Route53Configuration

为 API 配置 Route53 记录集。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
DistributionDomainName: String
EvaluateTargetHealth: Boolean
HostedZoneId: String
HostedZoneName: String
```

```
  IpV6: Boolean
  Region: String
  SetIdentifier: String
  VpcEndpointDomainName: String
  VpcEndpointHostedZoneId: String
```

属性

`DistributionDomainName`

配置 API 自定义域名的自定义分配。

类型：字符串

必需：否

默认：使用 API Gateway 分配。

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Route53::RecordSetGroup` `AliasTarget`资源的`DNSName`属性。

其他说明：CloudFront分配的域名。

`EvaluateTargetHealth`

如果 EvaluateTargetHealth 为 true，则别名记录将继承引用 Amazon 资源的运行状况，例如 Elastic Load Balancing 负载均衡器或托管区域中的其他记录。

类型：布尔值

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Route53::RecordSetGroup` `AliasTarget`资源的`EvaluateTargetHealth`属性。

其他说明：当别名目标为 CloudFront 分布时，您不能 EvaluateTargetHealth 将其设置为 true。

`HostedZoneId`

要在其中创建记录的托管区的 ID。

指定 `HostedZoneName` 或 `HostedZoneId`，但不能同时指定两者。如果您拥有多个使用相同域名的托管区域，则必须使用 `HostedZoneId` 指定托管区。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Route53::RecordSetGroup` `RecordSet`资源的`HostedZoneId`属性。

`HostedZoneName`

要在其中创建记录的托管区的名称。

指定 `HostedZoneName` 或 `HostedZoneId`，但不能同时指定两者。如果您拥有多个使用相同域名的托管区域，则必须使用 `HostedZoneId` 指定托管区。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Route53::RecordSetGroup` `RecordSet`资源的`HostedZoneName`属性。

`IpV6`

设置此属性后，将 Amazon SAM 创建一个`AWS::Route53::RecordSet`资源并将提供的资源的 `Type` 设置`AAAA`为 HostedZone。

类型：布尔值

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`Region`

仅限基于延迟的资源记录集：您创建此资源记录集所引用的资源的亚马逊 EC2 区域。资源通常是 Amazon 资源，例如 EC2 实例或 ELB 负载均衡器，并由 IP 地址或 DNS 域名引用，具体取决于记录类型。

当 Amazon Route 53 收到针对您已为其创建延迟资源记录集的域名和类型的 DNS 查询时，Route 53 会选择最终用户与关联亚马逊 EC2 地区之间延迟最低的延迟资源记录集。然后，Route 53 会返回与所选资源记录集相关的值。

请注意以下几点：

- 您只能为每个延迟资源记录集指定一个 `ResourceRecord`。
- 您只能为每个 Amazon EC2 地区创建一个延迟资源记录集。

- 您无需为所有 Amazon EC2 区域创建延迟资源记录集。Route 53 会从您已创建延迟资源记录集的区域中选择延迟性能最佳的区域。
- 您不能创建 `Name` 和 `Type` 元素的值与延迟资源记录集相同的非延迟资源记录集。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Route53::RecordSetGroupRecordSet`数据类型的 `Region`属性。

`SetIdentifier`

具有简单策略以外的路由策略的资源记录集：用于区分具有相同名称和类型组合的多个资源记录集的标识符，如名为 acme.example.com 且类型为 A 的多个加权资源记录集。在一组具有相同名称和类型的资源记录集中，每个资源记录集的 `SetIdentifier` 值必须唯一。

有关路由策略的信息，请参阅《Amazon Route 53 开发人员指南》中的选择路由策略。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Route53::RecordSetGroupRecordSet`数据类型的 `SetIdentifier`属性。

`VpcEndpointDomainName`

与 API Gateway VPC 服务关联的 VPC 接口终端节点的 DNS 名称。只有私有域名才需要此属性。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Route53::RecordSetAliasTarget`字段的`DNSName`属性。

`VpcEndpointHostedZoneId`

与 API Gateway VPC 服务关联的 VPC 接口终端节点的托管区域 ID。只有私有域名才需要此属性。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Route53::RecordSetAliasTarget`字段的`HostedZoneId`属性。

示例

基本示例

在此示例中，我们为 API 配置了自定义域和 Route 53 记录集。

YAML

```
Resources:
  MyApi:
    Type: AWS::Serverless::Api
    Properties:
      StageName: Prod
      Domain:
        DomainName: www.example.com
        CertificateArn: arn:aws:acm:us-east-1:123456789012:certificate/
abcdef12-3456-7890-abcd-ef1234567890
        EndpointConfiguration: REGIONAL
        Route53:
          HostedZoneId: ABCDEFGHIJKLMNOP
```

EndpointConfiguration

REST API 的端点类型。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
  Type: String
  VPCEndpointIds: List
```

属性

`Type`

REST API 的端点类型。

有效值：EDGE 或 REGIONAL 或 PRIVATE

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给AWS::ApiGateway::RestApiEndpointConfiguration数据类型的Types属性。

VPCEndpointIds

用于创建 Route53 IDs 别名的 REST API 的 VPC 终端节点列表。

类型：列表

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给AWS::ApiGateway::RestApiEndpointConfiguration数据类型的VpcEndpointIds属性。

示例

EndpointConfiguration

端点配置示例

YAML

```
EndpointConfiguration:
  Type: PRIVATE
  VPCEndpointIds:
    - vpce-123a123a
    - vpce-321a321a
```

AWS::Serverless::Application

将来自 Amazon Serverless Application Repository 或来自 Amazon S3 存储桶的无服务器应用程序嵌入为嵌套应用程序。嵌套应用程序以嵌套方式部署 AWS::CloudFormation::Stack资源，其中可以包含多种其他资源，包括其他AWS::Serverless::Application资源。

> **Note**
>
> 部署到时 Amazon CloudFormation， Amazon SAM 会将您的 Amazon SAM 资源转换为 Amazon CloudFormation 资源。有关更多信息，请参阅 生成的 Amazon CloudFormation 资源用于 Amazon SAM。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
Type: AWS::Serverless::Application
Properties:
  Location: String | ApplicationLocationObject
  NotificationARNs: List
  Parameters: Map
  Tags: Map
  TimeoutInMinutes: Integer
```

属性

`Location`

嵌套应用程序的模板 URL、文件路径或位置对象。

如果提供了模板 URL，则它必须遵循CloudFormation TemplateUrl 文档中指定的格式并包含有效的 CloudFormation 或 SAM 模板。可以使用 ApplicationLocationObject 指定已发布到 Amazon Serverless Application Repository 的应用程序。

如果提供了本地文件路径，则模板必须经过包含 `sam deploy` 或 `sam package` 命令的工作流程，才能让应用程序正确转换。

类型：字符串 | ApplicationLocationObject

必需：是

Amazon CloudFormation 兼容性：此属性类似于`AWS::CloudFormation::Stack`资源的`TemplateURL`属性。该 CloudFormation 版本无需使用ApplicationLocationObject即可从中检索应用程序 Amazon Serverless Application Repository。

`NotificationARNs`

向其发送堆栈事件通知的现有 Amazon SNS 主题的列表。

类型：列表

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::CloudFormation::Stack`资源的`NotificationARNs`属性。

`Parameters`

应用程序参数值。

类型：映射

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::CloudFormation::Stack`资源的`Parameters`属性。

`Tags`

指定要添加到此应用程序的标签的映射（字符串到字符串）。键和值只能包含字母数字字符。键的长度可以在 1 到 127 个 Unicode 字符之间，并且不能带有前缀“aws:”。值的长度可以在 1 到 255 个 Unicode 字符之间。

类型：映射

必需：否

Amazon CloudFormation 兼容性：此属性类似于`AWS::CloudFormation::Stack`资源的`Tags`属性。SAM 中的 Tags 属性由 Key: Value CloudFormation 对组成；其中包含标签对象的列表。创建堆栈后，SAM 将自动向该应用程序添加 `lambda:createdBy:SAM` 标签。此外，如果此应用程序来自 Amazon Serverless Application Repository，那么 SAM 还将自动添加两个附加标签`serverlessrepo:applicationId:`*`ApplicationId`*和`serverlessrepo:semanticVersion:`*`S`*

`TimeoutInMinutes`

Amazon CloudFormation 等待嵌套堆栈达到`CREATE_COMPLETE`状态的时间长度（以分钟为单位）。默认值为无超时。当 Amazon CloudFormation 检测到嵌套堆栈已达

到CREATE_COMPLETE状态时，它会将嵌套堆栈资源标记为CREATE_COMPLETE在父堆栈中，并继续创建父堆栈。如果超时时间在嵌套堆栈到达之前到期CREATE_COMPLETE，则会将嵌套堆栈 Amazon CloudFormation 标记为失败并回滚嵌套堆栈和父堆栈。

类型：整数

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给AWS::CloudFormation::Stack资源的TimeoutInMinutes属性。

返回值

Ref

当向 Ref 内置函数提供此资源的逻辑 ID 时，将返回底层 AWS::CloudFormation::Stack 资源的资源名称。

有关使用 Ref 函数的更多信息，请参阅《Amazon CloudFormation 用户指南》中的 Ref。

Fn:: GetAtt

Fn::GetAtt 返回一个此类型指定属性的值。以下为可用属性和示例返回值。

有关使用 Fn::GetAtt 的更多信息，请参阅《Amazon CloudFormation 用户指南》中的 Fn::GetAtt。

Outputs.ApplicationOutputName

带有名称 *ApplicationOutputName* 的堆栈输出的值。

示例

SAR 应用程序

使用来自无服务器应用程序存储库的模板的应用程序

YAML

```
Type: AWS::Serverless::Application
```

```
Properties:
  Location:
    ApplicationId: 'arn:aws:serverlessrepo:us-east-1:012345678901:applications/my-application'
    SemanticVersion: 1.0.0
  Parameters:
    StringParameter: parameter-value
    IntegerParameter: 2
```

正常应用

来自 S3 网址的应用程序

YAML

```
Type: AWS::Serverless::Application
Properties:
  Location: https://s3.amazonaws.com/sam-s3-demo-bucket/template.yaml
```

ApplicationLocationObject

已发布到 Amazon Serverless Application Repository 的应用程序。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
  ApplicationId: String
  SemanticVersion: String
```

属性

`ApplicationId`

应用程序的 Amazon 资源名称（ARN）。

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`SemanticVersion`

应用程序的语义版本。

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

示例

my-application

应用程序位置对象示例

YAML

```
Location:
  ApplicationId: 'arn:aws:serverlessrepo:us-east-1:012345678901:applications/my-application'
  SemanticVersion: 1.0.0
```

AWS::Serverless::Connector

配置两种资源之间的权限。有关连接器的简介，请参阅使用 Amazon SAM 连接器管理资源权限。

有关生成的 Amazon CloudFormation 资源的更多信息，请参阅Amazon CloudFormation 指定时生成的资源 AWS::Serverless::Connector。

要提供有关连接器的反馈，请在serverless-application-model Amazon GitHub 存储库中提交新问题。

> **Note**
>
> 部署到时 Amazon CloudFormation， Amazon SAM 会将您的 Amazon SAM 资源转换为 Amazon CloudFormation 资源。有关更多信息，请参阅 生成的 Amazon CloudFormation 资源用于 Amazon SAM。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下任一语法。

> **Note**
>
> 对于大多数用例，我们建议使用嵌入式连接器语法。随着时间的推移，嵌入在源资源中使其更易于读取和维护。当您需要引用不在同一 Amazon SAM 模板中的源资源（例如嵌套堆栈中的资源或共享资源）时，请使用`AWS::Serverless::Connector`语法。

嵌入式连接器

```
<source-resource-logical-id>:
  Connectors:
    <connector-logical-id:
      Properties:
        Destination: ResourceReference | List of ResourceReference
        Permissions: List
        SourceReference: SourceReference
```

AWS::Serverless::Connector

```
Type: AWS::Serverless::Connector
Properties:
  Destination: ResourceReference | List of ResourceReference
  Permissions: List
  Source: ResourceReference
```

属性

`Destination`

目标资源。

类型:ResourceReference| 清单 ResourceReference

必需：是

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`Permissions`

允许源资源对目标资源执行的权限类型。

`Read`包括允许从资源中读取数据的 Amazon Identity and Access Management (IAM) 操作。

`Write` 包括允许初始化和向资源写入数据的 IAM 操作。

有效值：`Read` 或 `Write`

类型：列表

必需：是

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`Source`

源资源。使用 `AWS::Serverless::Connector` 语法时为必需。

类型：ResourceReference

必需：条件

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`SourceReference`

源资源。

> **Note**
>
> 为源资源定义其他属性时，使用嵌入式连接器语法。

类型：SourceReference

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

示例

嵌入式连接器

以下示例使用嵌入式连接器定义 Amazon Lambda 函数和 Amazon DynamoDB 表之间的 `Write` 数据连接：

```
Transform: AWS::Serverless-2016-10-31
...
Resources:
  MyTable:
    Type: AWS::Serverless::SimpleTable
  MyFunction:
    Type: AWS::Serverless::Function
    Connectors:
      MyConn:
        Properties:
          Destination:
            Id: MyTable
          Permissions:
            - Write
    ...
```

以下示例使用嵌入式连接器定义 `Read` 和 `Write` 权限：

```
Transform: AWS::Serverless-2016-10-31
...
Resources:
  MyFunction:
    Type: AWS::Serverless::Function
    Connectors:
      MyConn:
        Properties:
          Destination:
            Id: MyTable
          Permissions:
            - Read
            - Write
  MyTable:
    Type: AWS::DynamoDB::Table
    ...
```

以下示例使用嵌入式连接器定义具有 `Id` 以外属性的源资源：

```
Transform: AWS::Serverless-2016-10-31
Transform: AWS::Serverless-2016-10-31
...
Resources:
  MyApi:
    Type: AWS::Serverless::Api
    Connectors:
      ApitoLambdaConn:
        Properties:
          SourceReference:
            Qualifier: Prod/GET/foobar
          Destination:
            Id: MyTable
          Permissions:
            - Read
            - Write
  MyTable:
    Type: AWS::DynamoDB::Table
    ...
```

AWS::Serverless::Connector

以下示例使用该AWS::Serverless::Connector资源对一个 Amazon Lambda 函数 Amazon DynamoDB 表进行读取和写入：

```
MyConnector:
  Type: AWS::Serverless::Connector
  Properties:
    Source:
      Id: MyFunction
    Destination:
      Id: MyTable
    Permissions:
      - Read
      - Write
```

以下示例使用 AWS::Serverless::Connector 资源让 Lambda 函数写入 Amazon SNS 主题，两个资源位于同一个模板中：

```
MyConnector:
  Type: AWS::Serverless::Connector
  Properties:
```

```
    Source:
      Id: MyLambda
    Destination:
      Id: MySNSTopic
    Permissions:
      - Write
```

以下示例使用 AWS::Serverless::Connector 资源让 Amazon SNS 主题写入 Lambda 函数，然后该函数写入 Amazon DynamoDB 表，所有资源都位于同一个模板中：

```
Transform: AWS::Serverless-2016-10-31
Resources:
  Topic:
    Type: AWS::SNS::Topic
    Properties:
      Subscription:
        - Endpoint: !GetAtt Function.Arn
          Protocol: lambda

  Function:
    Type: AWS::Serverless::Function
    Properties:
      Runtime: nodejs16.x
      Handler: index.handler
      InlineCode: |
        const AWS = require('aws-sdk');
        exports.handler = async (event, context) => {
          const docClient = new AWS.DynamoDB.DocumentClient();
          await docClient.put({
            TableName: process.env.TABLE_NAME,
            Item: {
              id: context.awsRequestId,
              event: JSON.stringify(event)
            }
          }).promise();
        };
      Environment:
        Variables:
          TABLE_NAME: !Ref Table

  Table:
    Type: AWS::Serverless::SimpleTable
```

```
  TopicToFunctionConnector:
    Type: AWS::Serverless::Connector
    Properties:
      Source:
        Id: Topic
      Destination:
        Id: Function
      Permissions:
        - Write

  FunctionToTableConnector:
    Type: AWS::Serverless::Connector
    Properties:
      Source:
        Id: Function
      Destination:
        Id: Table
      Permissions:
        - Write
```

以下是上面示例中转换后的 Amazon CloudFormation 模板：

```
"FunctionToTableConnectorPolicy": {
  "Type": "AWS::IAM::ManagedPolicy",
  "Metadata": {
    "aws:sam:connectors": {
      "FunctionToTableConnector": {
        "Source": {
          "Type": "AWS::Lambda::Function"
        },
        "Destination": {
          "Type": "AWS::DynamoDB::Table"
        }
      }
    }
  },
  "Properties": {
    "PolicyDocument": {
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Action": [
```

```
            "dynamodb:PutItem",
            "dynamodb:UpdateItem",
            "dynamodb:DeleteItem",
            "dynamodb:BatchWriteItem",
            "dynamodb:PartiQLDelete",
            "dynamodb:PartiQLInsert",
            "dynamodb:PartiQLUpdate"
          ],
          "Resource": [
            {
              "Fn::GetAtt": [
                "MyTable",
                "Arn"
              ]
            },
            {
              "Fn::Sub": [
                "${DestinationArn}/index/*",
                {
                  "DestinationArn": {
                    "Fn::GetAtt": [
                      "MyTable",
                      "Arn"
                    ]
                  }
                }
              ]
            }
          ]
        }
      ]
    },
    "Roles": [
      {
        "Ref": "MyFunctionRole"
      }
    ]
  }
}
```

ResourceReference

对 AWS::Serverless::Connector 资源类型使用的资源的引用。

> **Note**
>
> 对于同一模板中的资源，请提供 `Id`。对于不在同一模板中的资源，请使用其他属性的组合。有关更多信息，请参阅 Amazon SAM 连接器参考。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```yaml
Arn: String
Id: String
Name: String
Qualifier: String
QueueUrl: String
ResourceId: String
RoleName: String
Type: String
```

属性

`Arn`

资源的 ARN。

类型：字符串

必需：条件

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`Id`

同一模板中资源的逻辑 ID。

> **Note**
>
> 指定后`Id`，如果连接器生成 Amazon Identity and Access Management (IAM) 策略，则将从资源`Id`中推断出与这些策略关联的 IAM 角色。如果 `Id` 未指定，则为连接器提供资源 `RoleName`，以便将生成的 IAM 策略附加到 IAM 角色。

类型：字符串

必需：条件

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`Name`

资源的名称。

类型：字符串

必需：条件

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`Qualifier`

缩小其范围的资源限定符。`Qualifier` 替换资源约束 ARN 末尾的 `*` 值。有关示例，请参阅API Gateway 调用 Lambda 函数。

> **Note**
>
> 限定符定义因资源类型而异。有关受支持的源资源和目的地资源类型的列表，请参阅Amazon SAM 连接器参考。

类型：字符串

必需：条件

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`QueueUrl`

Amazon SQS 队列 URL。此属性仅适用于 Amazon SQS 资源。

类型：字符串

必需：条件

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`ResourceId`

资源的 ID。例如，API Gateway API ID。

类型：字符串

必需：条件

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`RoleName`

与资源关联的角色名称。

> **Note**
>
> 指定 `Id` 后，如果连接器生成 IAM 策略，则将从资源 `Id` 中推断出与这些策略关联的 IAM 角色。如果 `Id` 未指定，则为连接器提供资源 `RoleName`，以便将生成的 IAM 策略附加到 IAM 角色。

类型：字符串

必需：条件

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`Type`

资源的 Amazon CloudFormation 类型。有关更多信息，请参阅Amazon 资源和属性类型参考。

类型：字符串

必需：条件

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

示例

API Gateway 调用 Lambda 函数

以下示例使用该AWS::Serverless::Connector资源允许 Amazon API Gateway 调用 Amazon Lambda 函数。

YAML

```
Transform: AWS::Serverless-2016-10-31
Resources:
  MyRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Statement:
          - Effect: Allow
            Action: sts:AssumeRole
            Principal:
              Service: lambda.amazonaws.com
      ManagedPolicyArns:
        - !Sub arn:${AWS::Partition}:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole

  MyFunction:
    Type: AWS::Lambda::Function
    Properties:
      Role: !GetAtt MyRole.Arn
      Runtime: nodejs16.x
      Handler: index.handler
      Code:
        ZipFile: |
          exports.handler = async (event) => {
            return {
              statusCode: 200,
              body: JSON.stringify({
                "message": "It works!"
              }),
```

```
            };
          };

  MyApi:
    Type: AWS::ApiGatewayV2::Api
    Properties:
      Name: MyApi
      ProtocolType: HTTP

  MyStage:
    Type: AWS::ApiGatewayV2::Stage
    Properties:
      ApiId: !Ref MyApi
      StageName: prod
      AutoDeploy: True

  MyIntegration:
    Type: AWS::ApiGatewayV2::Integration
    Properties:
      ApiId: !Ref MyApi
      IntegrationType: AWS_PROXY
      IntegrationUri: !Sub arn:aws:apigateway:${AWS::Region}:lambda:path/2015-03-31/
functions/${MyFunction.Arn}/invocations
      IntegrationMethod: POST
      PayloadFormatVersion: "2.0"

  MyRoute:
    Type: AWS::ApiGatewayV2::Route
    Properties:
      ApiId: !Ref MyApi
      RouteKey: GET /hello
      Target: !Sub integrations/${MyIntegration}

  MyConnector:
    Type: AWS::Serverless::Connector
    Properties:
      Source: # Use 'Id' when resource is in the same template
        Type: AWS::ApiGatewayV2::Api
        ResourceId: !Ref MyApi
        Qualifier: prod/GET/hello # Or "*" to allow all routes
      Destination: # Use 'Id' when resource is in the same template
        Type: AWS::Lambda::Function
        Arn: !GetAtt MyFunction.Arn
      Permissions:
```

```
        - Write

Outputs:
  Endpoint:
    Value: !Sub https://${MyApi}.execute-api.${AWS::Region}.${AWS::URLSuffix}/prod/
hello
```

SourceReference

对 AWS::Serverless::Connector 资源类型使用的源资源的引用。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
Qualifier: String
```

属性

`Qualifier`

缩小其范围的资源限定符。`Qualifier` 替换资源约束 ARN 末尾的 `*` 值。

> **Note**
> 限定符定义因资源类型而异。有关受支持的源资源和目的地资源类型的列表，请参阅 Amazon SAM 连接器参考。

类型：字符串

必需：条件

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

示例

以下示例使用嵌入式连接器定义具有 **`Id`** 以外属性的源资源：

```
Transform: AWS::Serverless-2016-10-31
...
Resources:
  MyApi:
    Type: AWS::Serverless::Api
    Connectors:
      ApitoLambdaConn:
        Properties:
          SourceReference:
            Qualifier: Prod/GET/foobar
          Destination:
            Id: MyTable
          Permissions:
            - Read
            - Write
  MyTable:
    Type: AWS::DynamoDB::Table
    ...
```

AWS::Serverless::Function

创建触发该 Amazon Lambda 函数的函数、 Amazon Identity and Access Management (IAM) 执行角色和事件源映射。

该AWS::Serverless::Function资源还支持 r `Metadata` esource 属性，因此您可以指示 Amazon SAM 构建应用程序所需的自定义运行时。有关构建自定义运行时系统的更多信息，请参阅 在中使用自定义运行时构建 Lambda 函数 Amazon SAM。

> **Note**
>
> 部署到时 Amazon CloudFormation， Amazon SAM 会将您的 Amazon SAM 资源转换为 Amazon CloudFormation 资源。有关更多信息，请参阅 生成的 Amazon CloudFormation 资源用于 Amazon SAM。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
Type: AWS::Serverless::Function
```

```
Properties:
  Architectures: List
  AssumeRolePolicyDocument: JSON
  AutoPublishAlias: String
  AutoPublishAliasAllProperties: Boolean
  AutoPublishCodeSha256: String
  CodeSigningConfigArn: String
  CodeUri: String | FunctionCode
  DeadLetterQueue: Map | DeadLetterQueue
  DeploymentPreference: DeploymentPreference
  Description: String
  Environment: Environment
  EphemeralStorage: EphemeralStorage
  EventInvokeConfig: EventInvokeConfiguration
  Events: EventSource
  FileSystemConfigs: List
  FunctionName: String
  FunctionUrlConfig: FunctionUrlConfig
  Handler: String
  ImageConfig: ImageConfig
  ImageUri: String
  InlineCode: String
  KmsKeyArn: String
  Layers: List
  LoggingConfig: LoggingConfig
  MemorySize: Integer
  PackageType: String
  PermissionsBoundary: String
  Policies: String | List | Map
  PropagateTags: Boolean
  ProvisionedConcurrencyConfig: ProvisionedConcurrencyConfig
  RecursiveLoop: String
  ReservedConcurrentExecutions: Integer
  Role: String
  RolePath: String
  Runtime: String
  RuntimeManagementConfig: RuntimeManagementConfig
  SnapStart: SnapStart
  SourceKMSKeyArn: String
  Tags: Map
  Timeout: Integer
  Tracing: String
  VersionDescription: String
```

```
  VpcConfig: VpcConfig
```

属性

`Architectures`

该函数的指令集架构。

有关此属性更多信息，请参阅《Amazon Lambda 开发人员指南》中的 Lambda 指令集架构。

有效值：`x86_64` 或 `arm64` 之一。

类型：列表

必需：否

默认值：`x86_64`

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::Function`资源的`Architectures`属性。

`AssumeRolePolicyDocument`

AssumeRolePolicyDocument 为该函数创建`Role`的默认值添加一个。如果未指定此属性，则为该函数 Amazon SAM 添加默认的代入角色。

类型：JSON

必需：否

Amazon CloudFormation 兼容性：此属性类似于`AWS::IAM::Role`资源的`AssumeRolePolicyDocument`属性。 Amazon SAM 将此属性添加到为此函数生成的 IAM 角色中。如果为此函数提供了角色的 Amazon 资源名称（ARN），则此属性将不执行任何操作。

`AutoPublishAlias`

Lambda 别名的名称。要了解有关 Lambda 别名的更多信息，请参阅《Amazon Lambda 开发人员指南》中的 Lambda 函数别名。有关使用此属性的示例，请参见 使用以下方法逐步部署无服务器应用程序 Amazon SAM。

Amazon SAM 生成 AWS::Lambda::Version 和 AWS::Lambda::Alias设置此属性时的资源。有关此场景的更多信息，请参阅AutoPublishAlias 属性已指定。有关生成的 Amazon CloudFormation 资源的一般信息，请参阅生成的 Amazon CloudFormation 资源用于 Amazon SAM。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`AutoPublishAliasAllProperties`

指定何时创建新的 `AWS::Lambda::Version`。当为 `true` 时，修改 Lambda 函数中的任何属性时，就会创建新的 Lambda 版本。当为 `false` 时，只有修改了以下任何属性时，才会创建新的 Lambda 版本：

- `Environment`、`MemorySize` 或者 `SnapStart`。
- 导致 `Code` 属性更新的任何更改，例如 `CodeDict`、`ImageUri`、或 `InlineCode`。

此属性需要定义 `AutoPublishAlias`。

如果也指定 `AutoPublishCodeSha256`，则其行为优先于 `AutoPublishAliasAllProperties: true`。

类型：布尔值

必需：否

默认值：`false`

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`AutoPublishCodeSha256`

使用时，此字符串将与 `CodeUri` 值一起确定是否需要发布新的 Lambda 版本。此属性通常用于解决以下部署问题：部署包存储在 Amazon S3 位置，且被包含更新后的 Lambda 函数代码的新部署包取代，但 `CodeUri` 属性保持不变（相反情况是新的部署包上传到新的 Amazon S3 位置并且 `CodeUri` 更改为新位置）。

此问题由具有以下特征的 Amazon SAM 模板标记：

- `DeploymentPreference` 对象配置为逐步部署（如使用以下方法逐步部署无服务器应用程序 Amazon SAM中所述）
- `AutoPublishAlias` 属性已设置，且在部署之间不会更改
- `CodeUri` 属性已设置，且在部署之间不会更改。

在这种情况下，更新 AutoPublishCodeSha256 会导致成功创建新的 Lambda 版本。但是，部署到 Amazon S3 的新函数代码将无法识别。要识别新的函数代码，请考虑在 Amazon S3 存储桶中使用版本控制。为 Lambda 函数指定 Version 属性，并将存储桶配置为始终使用最新的部署包。

在这种情况下，要成功触发逐步部署，必须为 AutoPublishCodeSha256 提供一个唯一的值。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

CodeSigningConfigArn

的 ARN AWS::Lambda::CodeSigningConfig资源，用于为此功能启用代码签名。有关代码签名的更多信息，请参阅 为您的 Amazon SAM 应用程序设置代码签名。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给AWS::Lambda::Function资源的CodeSigningConfigArn属性。

CodeUri

函数的代码。可接受的值包括：

- 该函数的 Amazon S3 URI。例如 s3://bucket-123456789/sam-app/1234567890abcdefg。
- 函数的本地路径。例如 hello_world/。
- 一个 FunctionCode 对象。

> **Note**
>
> 如果您提供函数的 Amazon S3 URI 或 FunctionCode 对象，则必须引用有效的 Lambda 部署包。
> 如果您提供了本地文件路径，请使用 Amazon SAM CLI 在部署时上传本地文件。要了解更多信息，请参阅如何在 Amazon SAM 部署时上传本地文件。
> 如果你在CodeUri属性中使用内部函数， Amazon SAM 将无法正确解析这些值。请考虑改用AWS::Language扩展程序转换。

类型：[字符串 | FunctionCode]

必填：条件性。当 `PackageType` 设置为 `Zip` 时，则 `CodeUri` 或 `InlineCode` 中的一种为必需。

Amazon CloudFormation 兼容性：此属性类似于`AWS::Lambda::Function`资源的 `Code`属性。嵌套的 Amazon S3 属性的命名有所不同。

`DeadLetterQueue`

在 Lambda 发送无法处理的事件时配置 Amazon Simple Notification Service (Amazon SNS)主题或 Amazon Simple Queue Service (Amazon SQS) 队列。有关死信队列功能的更多信息，请参阅《Amazon Lambda 开发人员指南》中的死信队列。

> **Note**
>
> 如果您的 Lambda 函数的事件源是 Amazon SQS 队列，请为源队列而不是 Lambda 函数配置死信队列。您为函数配置的死信队列用于函数的异步调用队列，而不是用于事件源队列。

类型：地图 | DeadLetterQueue

必需：否

Amazon CloudFormation 兼容性：此属性类似于`AWS::Lambda::Function`资源的`DeadLetterConfig`属性。在 Amazon CloudFormation 中，类型是派生自的`TargetArn`，而在中， Amazon SAM 你必须将类型与一起传递`TargetArn`。

`DeploymentPreference`

用于启用逐步 Lambda 部署的设置。

如果指定了`DeploymentPreference`对象，则 Amazon SAM 会创建一个 AWS::CodeDeploy::Application调用`ServerlessDeploymentApplication`（每个堆栈一个），一个 AWS::CodeDeploy::DeploymentGroup已调用*`<function-logical-id>`*`DeploymentGroup`，并且 AWS::IAM::Role被称为`CodeDeployServiceRole`。

类型：DeploymentPreference

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

另请参阅：有关此属性的更多信息，请参阅 使用以下方法逐步部署无服务器应用程序 Amazon SAM。

`Description`

该函数的描述。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::Function`资源的`Description`属性。

`Environment`

运行时系统环境的配置。

类型：环境

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::Function`资源的`Environment`属性。

`EphemeralStorage`

指定 `/tmp` 中 Lambda 函数可用的磁盘空间（以 MB 为单位）的对象。

有关此属性的更多信息，请参阅《Amazon Lambda 开发人员指南》中的 Lambda 执行环境。

类型：EphemeralStorage

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::Function`资源的`EphemeralStorage`属性。

`EventInvokeConfig`

描述 Lambda 函数的事件调用配置的对象。

类型：EventInvokeConfiguration

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`Events`

指定触发此函数的事件。事件由一个类型和一组依赖于该类型的属性组成。

类型：EventSource

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`FileSystemConfigs`

指定亚马逊弹性文件系统 (Amazon EFS) 文件系统的连接设置的FileSystemConfig对象列表。

如果您的模板包含 AWS::EFS::MountTargetresource，您还必须指定`DependsOn`资源属性，以确保在函数运行之前创建或更新装载目标。

类型：列表

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::Function`资源的`FileSystemConfigs`属性。

`FunctionName`

函数的名称。如果您没有指定名称，则系统为您生成唯一的名称。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::Function`资源的`FunctionName`属性。

`FunctionUrlConfig`

描述函数 URL 的对象。函数 URL 是可用于调用函数的 HTTPS 端点。

有关更多信息，请参阅《Amazon Lambda 开发者指南》URLs中的函数。

类型：FunctionUrlConfig

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`Handler`

代码中被调用以开始执行的函数。仅当属性 `PackageType` 为 `Zip` 时，该属性是必需属性。

类型：字符串

必需：条件

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::Function`资源的`Handler`属性。

`ImageConfig`

用于配置 Lambda 容器映像设置的对象。有关更多信息，请参阅《Amazon Lambda 开发人员指南》中的将容器映像与 Lambda 结合使用。

类型：ImageConfig

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::Function`资源的`ImageConfig`属性。

`ImageUri`

Lambda 函数容器映像的 Amazon Elastic Container Registry (Amazon ECR) 存储库 URI 仅当 `PackageType` 属性设置为 `Image` 时，此属性才适用，否则将被忽略。有关更多信息，请参阅《Amazon Lambda 开发人员指南》中的将容器映像与 Lambda 结合使用。

> **Note**
>
> 如果将该`PackageType`属性设置为`Image`，`ImageUri`则要么为必填项，要么必须使用 Amazon SAM 模板文件中的必要`Metadata`条目来构建应用程序。有关更多信息，请参阅默认版本使用 Amazon SAM。

使用必要的 `Metadata` 条目构建应用程序优先于 `ImageUri`，因此，如果您同时指定两者，则 `ImageUri` 会被忽略。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::FunctionCode`数据类型的`ImageUri`属性。

`InlineCode`

直接在模板中编写的 Lambda 函数代码。仅当 `PackageType` 属性设置为 `Zip` 时，此属性才适用，否则将被忽略。

> **Note**
>
> 如果 `PackageType` 属性设置为 `Zip`（默认），则 `CodeUri` 或 `InlineCode` 中的一个为必需。

类型：字符串

必需：条件

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::FunctionCode`数据类型的`ZipFile`属性。

`KmsKeyArn`

Lambda 用来加密和解密函数环境变量的 Amazon Key Management Service (Amazon KMS) 密钥的 ARN。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::Function`资源的`KmsKeyArn`属性。

`Layers`

此函数应使用的列表。`LayerVersion` ARNs 此处指定的顺序是运行 Lambda 函数时它们的导入顺序。该版本要么是包含版本的完整 ARN，要么是对资源的引用。 LayerVersion 例如，对的引用`LayerVersion`将是，`!Ref MyLayer`而包含版本的完整 ARN 将是。`arn:aws:lambda:`*`region`*`:`*`account-id`*`:layer:`*`layer-name`*`:`*`version`*

类型：列表

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::Function`资源的`Layers`属性。

`LoggingConfig`

该函数的 Amazon CloudWatch 日志配置设置。

类型：LoggingConfig

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::Function`资源的`LoggingConfig`属性。

`MemorySize`

每次调用函数时分配的内存大小（以 MB 为单位）。

类型：整数

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::Function`资源的`MemorySize`属性。

`PackageType`

Lambda 函数的部署包类型。有关更多信息，请参阅《Amazon Lambda 开发人员指南》中的Lambda 部署包。

备注：

1. 如果此属性设置为 `Zip`（默认），则将应用 `CodeUri` 或 `InlineCode`，并忽略 `ImageUri`。

2. 如果此属性设置为 `Image`，则仅应用 `ImageUri`，并忽略 `CodeUri` 和 `InlineCode`。存储函数容器镜像所需的 Amazon ECR 存储库可以由自动创建 Amazon SAM CLI。 有关更多信息，请参阅sam deploy。

有效值：`Zip` 或 `Image`

类型：字符串

必需：否

默认值：`Zip`

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::Function`资源的`PackageType`属性。

`PermissionsBoundary`

用于此函数执行角色的权限边界的 ARN。仅当为您生成角色时，此属性才起作用。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::IAM::Role`资源的`PermissionsBoundary`属性。

`Policies`

此函数的权限策略。策略将附加到函数的默认 Amazon Identity and Access Management (IAM) 执行角色中。

此属性接受单个值或值列表。允许的值包括：

- Amazon SAM策略模板.
- 这些区域有：ARN Amazon 托管策略或客户托管策略。
- 以下列表中 Amazon 托管策略的名称。
- 内联 IAM 策略的格式为 YAML 作为地图。

> **Note**
>
> 如果指定 `Role` 属性，则将忽略该属性。

类型：字符串 | 列表 | 映射

必需：否

Amazon CloudFormation 兼容性：此属性类似于`AWS::IAM::Role`资源的`Policies`属性。

`PropagateTags`

指明是否将 `Tags` 属性中的标签传递给 AWS::Serverless::Function 生成的资源。指定 `True` 以在生成的资源中传播标签。

类型：布尔值

必需：否

默认值：`False`

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`ProvisionedConcurrencyConfig`

函数别名的预置并发配置。

> **Note**
>
> 仅当设置 `AutoPublishAlias` 时才可以指定 `ProvisionedConcurrencyConfig`。否则将导致错误。

类型：ProvisionedConcurrencyConfig

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::Alias`资源的`ProvisionedConcurrencyConfig`属性。

`RecursiveLoop`

函数的递归循环检测配置的状态。

当此值设置为 `Allow` 且 Lambda 检测到您的函数作为递归循环的一部分调用时，它不会采取任何操作。

当此值设置为 `Terminate` 且 Lambda 检测到您的函数作为递归循环的一部分调用时，它会停止调用函数并通知您。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::Function`资源的`RecursiveLoop`属性。

`ReservedConcurrentExecutions`

希望为函数预留的最大并发执行数。

有关此属性更多信息，请参阅《Amazon Lambda 开发人员指南》中的 Lambda 函数扩展。

类型：整数

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::Function`资源的`ReservedConcurrentExecutions`属性。

`Role`

要用作此函数执行角色的 IAM 角色的 ARN。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性类似于`AWS::Lambda::Function`资源的`Role`属性。这在中是必需的 Amazon CloudFormation ，但不是必填的 Amazon SAM。如果未指定角色，则会为您创建一个逻辑 ID 为 `<function-logical-id>`Role 的角色。

`RolePath`

函数的 IAM 执行角色的路径。

生成角色时请使用此属性。当使用 `Role` 属性指定角色时，请勿使用。

类型：字符串

必需：条件

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::IAM::Role`资源的`Path`属性。

`Runtime`

函数的运行时的标识符。仅当属性 `PackageType` 为 `Zip` 时，该属性是必需属性。

> **Note**
>
> 如果为此属性指定标`provided`识符，则可以使用 res `Metadata` ource 属性 Amazon SAM 来指示构建此函数所需的自定义运行时。有关构建自定义运行时系统的更多信息，请参阅在中使用自定义运行时构建 Lambda 函数 Amazon SAM。

类型：字符串

必需：条件

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::Function`资源的`Runtime`属性。

`RuntimeManagementConfig`

为 Lambda 函数配置运行时系统管理选项，例如运行时环境更新、回滚行为以及选择特定的运行时版本。要了解更多信息，请参阅《Amazon Lambda 开发人员指南》中的 Lambda 运行时系统更新。

类型：RuntimeManagementConfig

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::Function`资源的`RuntimeManagementConfig`属性。

`SnapStart`

创建任何新 Lambda 函数版本的快照。快照是初始化函数的缓存状态，包括其所有依赖项。函数仅被初始化一次，缓存的状态将在未来的所有调用中重复使用，从而通过减少必须初始化函数的次数来提高应用程序性能。要了解更多信息，请参阅Amazon Lambda 开发人员指南SnapStart中的使用 Lambda 提高启动性能。

类型：SnapStart

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::Function`资源的`SnapStart`属性。

`SourceKmsKeyArn`

表示用于加密客户邮政编码的 KMS 密钥 ARN。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::FunctionCode`数据类型的`SourceKmsKeyArn`属性。

`Tags`

指定添加到此函数的标签的映射（字符串到字符串）。有关标签的有效键和值的详细信息，请参阅《Amazon Lambda 开发人员指南》中的标签键和值要求。

创建堆栈后， Amazon SAM 会自动向此 Lambda 函数以及为此函数生成的默认角色添加`lambda:createdBy:SAM`标签。

类型：映射

必需：否

Amazon CloudFormation 兼容性：此属性类似于`AWS::Lambda::Function`资源的`Tags`属性。中的`Tags`属性由键值对 Amazon SAM 组成（而 Amazon CloudFormation 在此属性中则由`Tag`对象列表组成）。此外，还 Amazon SAM 会自动向此 Lambda 函数以及为此函数生成的默认角色添加`lambda:createdBy:SAM`标签。

`Timeout`

函数在停止之前可以运行的最长时间（以秒为单位）。

类型：整数

必需：否

原定设置值：3

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::Function`资源的`Timeout`属性。

`Tracing`

指定函数的 X-Ray 跟踪模式的字符串。

- `Active` – 为该函数激活 X-Ray 跟踪
- `Disabled` – 为该函数停用 X-Ray 跟踪
- `PassThrough` – 为该函数激活 X-Ray 跟踪 采样决策委托给下游服务。

如果指定为 `Active` 或 `PassThrough` 且未设置 `Role` 属性，则 Amazon SAM 会将 `arn:aws:iam::aws:policy/AWSXrayWriteOnlyAccess` 策略添加到它为您创建的 Lambda 执行角色中。

有关 X-Ray 的更多信息，请参阅《Amazon Lambda 开发者指南》 Amazon X-Ray中的 “Amazon Lambda 与一起使用”。

有效值：[`Active`|`Disabled`|`PassThrough`]

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性类似于`AWS::Lambda::Function`资源的`TracingConfig`属性。

`VersionDescription`

指定在新的 Lambda 版本资源上添加的 `Description` 字段。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::Version`资源的`Description`属性。

`VpcConfig`

该配置使该函数能够访问您的虚拟私有云（VPC）中的私有资源。

类型：VpcConfig

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::Function`资源的`VpcConfig`属性。

返回值

Ref

当向 `Ref` 内置函数提供此资源的逻辑 ID 时，将返回底层 Lambda 函数的资源名称。

有关使用 `Ref` 函数的更多信息，请参阅《Amazon CloudFormation 用户指南》中的 `Ref`。

Fn:: GetAtt

`Fn::GetAtt` 返回一个此类型指定属性的值。以下为可用属性和示例返回值。

有关使用 `Fn::GetAtt` 的更多信息，请参阅《Amazon CloudFormation 用户指南》中的 `Fn::GetAtt`。

`Arn`

底层 Lambda 函数的 ARN。

示例

简单函数

以下是 Amazon S3 存储桶中包类型为 `Zip`（默认）的 AWS::Serverless::Function 资源和函数代码的基本示例。

YAML

```
Type: AWS::Serverless::Function
Properties:
  Handler: index.handler
  Runtime: python3.9
  CodeUri: s3://bucket-name/key-name
```

函数属性示例

以下是使用 `InlineCode`、`Layers`、`Tracing`、`Policies`、`Amazon EFS`和 `Api` 事件源的包类型为 `Zip`（默认）的 AWS::Serverless::Function 示例。

YAML

```
Type: AWS::Serverless::Function
DependsOn: MyMountTarget        # This is needed if an AWS::EFS::MountTarget resource
 is declared for EFS
Properties:
  Handler: index.handler
  Runtime: python3.9
  InlineCode: |
    def handler(event, context):
      print("Hello, world!")
  ReservedConcurrentExecutions: 30
  Layers:
    - Ref: MyLayer
  Tracing: Active
  Timeout: 120
  FileSystemConfigs:
    - Arn: !Ref MyEfsFileSystem
      LocalMountPath: /mnt/EFS
  Policies:
    - AWSLambdaExecute
    - Version: '2012-10-17'
      Statement:
```

```
        - Effect: Allow
          Action:
            - s3:GetObject
            - s3:GetObjectACL
          Resource: 'arn:aws:s3:::sam-s3-demo-bucket/*'
  Events:
    ApiEvent:
      Type: Api
      Properties:
        Path: /path
        Method: get
```

ImageConfig 示例

以下是包类型为 `Image` 的 Lambda 函数的 `ImageConfig` 示例。

YAML

```
HelloWorldFunction:
  Type: AWS::Serverless::Function
  Properties:
    PackageType: Image
    ImageUri: account-id.dkr.ecr.region.amazonaws.com/ecr-repo-name:image-name
    ImageConfig:
      Command:
        - "app.lambda_handler"
      EntryPoint:
        - "entrypoint1"
      WorkingDirectory: "workDir"
```

RuntimeManagementConfig 例子

配置为根据当前行为更新其运行时环境的 Lambda 函数：

```
TestFunction
  Type: AWS::Serverless::Function
  Properties:
    ...
    Runtime: python3.9
    RuntimeManagementConfig:
      UpdateRuntimeOn: Auto
```

配置为在函数更新时更新其运行时环境的 Lambda 函数：

```
TestFunction
  Type: AWS::Serverless::Function
  Properties:
    ...
    Runtime: python3.9
    RuntimeManagementConfig:
      UpdateRuntimeOn: FunctionUpdate
```

配置为手动更新其运行时环境的 Lambda 函数：

```
TestFunction
  Type: AWS::Serverless::Function
  Properties:
    ...
    Runtime: python3.9
    RuntimeManagementConfig:
      RuntimeVersionArn: arn:aws:lambda:us-
east-1::runtime:4c459dd0104ee29ec65dcad056c0b3ddbe20d6db76b265ade7eda9a066859b1e
      UpdateRuntimeOn: Manual
```

SnapStart 示例

在未来版本中 SnapStart 启用的 Lambda 函数示例：

```
TestFunc
  Type: AWS::Serverless::Function
  Properties:
    ...
    SnapStart:
      ApplyOn: PublishedVersions
```

DeadLetterQueue

指定 (Amazon Lambda Lambda) 在无法处理事件时向其发送事件的 SQS 队列或 SNS 主题。有关死信队列功能的更多信息，请参阅《Amazon Lambda 开发人员指南》中的死信队列。

SAM 将自动为您的 Lambda 函数执行角色添加相应的权限，以授予 Lambda 服务访问资源的权限。将为 SQS 队列添加 sqs:，SendMessage 将为 SNS 主题添加 SNS: Publish。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
  TargetArn: String
  Type: String
```

属性

`TargetArn`

Amazon SQS 队列或 Amazon SNS 主题的 Amazon 资源名称（ARN）。

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::FunctionDeadLetterConfig`数据类型的`TargetArn`属性。

`Type`

死信队列的类型。

有效值：`SNS`、`SQS`

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

示例

DeadLetterQueue

SNS 主题的死信队列示例。

YAML

```
DeadLetterQueue:
  Type: SNS
```

```
TargetArn: arn:aws:sns:us-east-2:123456789012:my-topic
```

DeploymentPreference

指定启用逐步 Lambda 部署的配置。有关配置逐步 Lambda 部署的更多信息，请参阅 使用以下方法逐步部署无服务器应用程序 Amazon SAM。

> **Note**
>
> 要使用 `DeploymentPreference` 对象，必须在 AWS::Serverless::Function 中指定 `AutoPublishAlias`，否则将导致错误。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
Alarms: List
Enabled: Boolean
Hooks: Hooks
PassthroughCondition: Boolean
Role: String
TriggerConfigurations: List
Type: String
```

属性

`Alarms`

您希望由部署引发的任何错误触发的 CloudWatch 警报列表。

此属性接受 `Fn::If` 内置函数。有关使用 `Fn::If` 的示例模板，请参阅本主题底部的“示例”部分。

类型：列表

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`Enabled`

是否启用此部署首选项。

类型：布尔值

必需：否

默认值：True

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`Hooks`

流量转移之前和之后运行的验证 Lambda 函数。

类型：钩子

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`PassthroughCondition`

如果为 True，并且启用了此部署首选项，则函数的 Condition 将传递给生成的 CodeDeploy 资源。通常，应将其设置为 True。否则，即使函数的条件解析为 False，也会创建 CodeDeploy 资源。

类型：布尔值

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`Role`

用于流量转移的 IAM 角色 ARN。 CodeDeploy 如有提供，则不会创建 IAM 角色。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`TriggerConfigurations`

要与部署组关联的触发器配置列表。用于通知有关生命周期事件的 SNS 主题。

类型：列表

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::CodeDeploy::DeploymentGroup`资源的`TriggerConfigurations`属性。

`Type`

目前有两类部署类型：线性和金丝雀。有关可用部署类型的更多信息，请参阅 使用以下方法逐步部署无服务器应用程序 Amazon SAM。

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

示例

DeploymentPreference 带有交通前和交通后的挂钩。

包含流量前后钩子的部署首选项示例。

YAML

```
DeploymentPreference:
  Enabled: true
  Type: Canary10Percent10Minutes
  Alarms:
    - !Ref: AliasErrorMetricGreaterThanZeroAlarm
    - !Ref: LatestVersionErrorMetricGreaterThanZeroAlarm
  Hooks:
    PreTraffic:
      !Ref: PreTrafficLambdaFunction
    PostTraffic:
      !Ref: PostTrafficLambdaFunction
```

DeploymentPreference 使用 Fn:: If 内部函数

使用 `Fn::If` 配置警报的部署首选项示例。在本示例中，如果 `MyCondition` 为 `true`，则将配置 `Alarm1`；如果 `MyCondition` 为 `false`，则将配置 `Alarm2` 和 `Alarm5`。

YAML

```
DeploymentPreference:
  Enabled: true
  Type: Canary10Percent10Minutes
  Alarms:
    Fn::If:
      - MyCondition
      - - Alarm1
      - - Alarm2
        - Alarm5
```

Hooks

流量转移之前和之后运行的验证 Lambda 函数。

> **Note**
>
> 此属性中引用的 Lambda 函数配置结果的`CodeDeployLambdaAliasUpdate`对象 AWS::Lambda::Alias资源。有关更多信息，请参阅《Amazon CloudFormation 用户指南》中的CodeDeployLambdaAliasUpdate 策略。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
  PostTraffic: String
  PreTraffic: String
```

属性

`PostTraffic`

流量转移之后运行的 Lambda 函数。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`PreTraffic`

流量转移之前运行的 Lambda 函数。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

示例

钩子

示例钩子函数

YAML

```
Hooks:
  PreTraffic:
    Ref: PreTrafficLambdaFunction
  PostTraffic:
    Ref: PostTrafficLambdaFunction
```

EventInvokeConfiguration

异步 Lambda 别名或版本调用的配置选项。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
  DestinationConfig: EventInvokeDestinationConfiguration
```

```
  MaximumEventAgeInSeconds: Integer
  MaximumRetryAttempts: Integer
```

属性

DestinationConfig

一个配置对象，用于在 Lambda 处理事件后指定事件目的地。

类型：EventInvokeDestinationConfiguration

必需：否

Amazon CloudFormation 兼容性：此属性类似于`AWS::Lambda::EventInvokeConfig`资源的`DestinationConfig`属性。SAM 需要一个在中不存在的额外参数 “类型” CloudFormation。

MaximumEventAgeInSeconds

Lambda 发送到函数以进行处理的请求的最长期限。

类型：整数

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::EventInvokeConfig`资源的`MaximumEventAgeInSeconds`属性。

MaximumRetryAttempts

在函数返回错误前重试的最大次数。

类型：整数

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::EventInvokeConfig`资源的`MaximumRetryAttempts`属性。

示例

MaximumEventAgeInSeconds

MaximumEventAgeInSeconds 示例

YAML

```
EventInvokeConfig:
  MaximumEventAgeInSeconds: 60
  MaximumRetryAttempts: 2
  DestinationConfig:
    OnSuccess:
      Type: SQS
      Destination: arn:aws:sqs:us-west-2:012345678901:my-queue
    OnFailure:
      Type: Lambda
      Destination: !GetAtt DestinationLambda.Arn
```

EventInvokeDestinationConfiguration

一个配置对象，用于在 Lambda 处理事件后指定事件目的地。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
  OnFailure: OnFailure
  OnSuccess: OnSuccess
```

属性

`OnFailure`

处理失败的事件的目的地。

类型：OnFailure

必需：否

Amazon CloudFormation 兼容性：此属性类似于`AWS::Lambda::EventInvokeConfig`资源的`OnFailure`属性。需要 Type（这是仅适用于 SAM 的附加属性）。

`OnSuccess`

已成功处理的事件的目的地。

类型：OnSuccess

必需：否

Amazon CloudFormation 兼容性：此属性类似于`AWS::Lambda::EventInvokeConfig`资源的`OnSuccess`属性。需要 `Type`（这是仅适用于 SAM 的附加属性）。

示例

OnSuccess

OnSuccess 示例

YAML

```
EventInvokeConfig:
  DestinationConfig:
    OnSuccess:
      Type: SQS
      Destination: arn:aws:sqs:us-west-2:012345678901:my-queue
    OnFailure:
      Type: Lambda
      Destination: !GetAtt DestinationLambda.Arn
```

OnFailure

处理失败的事件的目的地。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
  Destination: String
  Type: String
```

属性

`Destination`

目标资源的 Amazon Resource Name (ARN)。

类型：字符串

必需：条件

Amazon CloudFormation 兼容性：此属性类似于`AWS::Lambda::EventInvokeConfig`资源的`OnFailure`属性。SAM 将向与该函数关联的自动生成的 IAM 角色添加任何必要的权限，以访问此属性中引用的资源。

其他说明：如果类型为 Lambda/EventBridge，则必须填写目的地。

`Type`

目标中引用的资源类型。支持的类型有`SQS``SNS`、`S3`、`Lambda`、和`EventBridge`。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

附加说明：如果类型为 SQS/SNS 且 `Destination` 属性留空，则 SQS/SNS 资源将由 SAM 自动生成。要引用该资源，请使用 `<function-logical-id>.DestinationQueue`（对于 SQS）或 `<function-logical-id>.DestinationTopic`（对于 SQS）。如果类型为 Lambda/EventBridge，`Destination`则为必填项。

示例

EventInvoke 包含 SQS 和 Lambda 目标的配置示例

在此示例中，没有给出 SQS OnSuccess 配置的目标，因此 SAM 隐式创建了一个 SQS 队列并添加了所有必要的权限。同样在本示例中，在 OnFailure 配置中指定了在模板文件中声明的 Lambda 资源的目标，因此 SAM 向此 Lambda 函数添加了调用目标 Lambda 函数所需的权限。

YAML

```
EventInvokeConfig:
  DestinationConfig:
    OnSuccess:
      Type: SQS
    OnFailure:
      Type: Lambda
      Destination: !GetAtt DestinationLambda.Arn  # Arn of a Lambda function declared
 in the template file.
```

EventInvoke 带有 SNS 目标的配置示例

在此示例中，为在 OnSuccess 配置的模板文件中声明的 SNS 主题提供了一个目标。

YAML

```
EventInvokeConfig:
  DestinationConfig:
    OnSuccess:
      Type: SNS
      Destination:
        Ref: DestinationSNS       # Arn of an SNS topic declared in the tempate file
```

OnSuccess

已成功处理的事件的目的地。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
  Destination: String
  Type: String
```

属性

`Destination`

目标资源的 Amazon 资源名称（ARN）。

类型：字符串

必需：条件

Amazon CloudFormation 兼容性：此属性类似于`AWS::Lambda::EventInvokeConfig`资源的`OnSuccess`属性。SAM 将向与该函数关联的自动生成的 IAM 角色添加任何必要的权限，以访问此属性中引用的资源。

其他说明：如果类型为 Lambda/EventBridge，则必须填写目的地。

`Type`

目标中引用的资源类型。支持的类型有`SQS``SNS`、`S3`、`Lambda`、和`EventBridge`。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

附加说明：如果类型为 SQS/SNS 且 `Destination` 属性留空，则 SQS/SNS 资源将由 SAM 自动生成。要引用该资源，请使用 `<function-logical-id>.DestinationQueue`（对于 SQS）或 `<function-logical-id>.DestinationTopic`（对于 SQS）。如果类型为 Lambda/EventBridge，`Destination`则为必填项。

示例

EventInvoke 包含 SQS 和 Lambda 目标的配置示例

在此示例中，没有给出 SQS OnSuccess 配置的目标，因此 SAM 隐式创建了一个 SQS 队列并添加了所有必要的权限。同样在本示例中，在 OnFailure 配置中指定了在模板文件中声明的 Lambda 资源的目标，因此 SAM 向此 Lambda 函数添加了调用目标 Lambda 函数所需的权限。

YAML

```
EventInvokeConfig:
  DestinationConfig:
    OnSuccess:
      Type: SQS
    OnFailure:
      Type: Lambda
      Destination: !GetAtt DestinationLambda.Arn  # Arn of a Lambda function declared
 in the template file.
```

EventInvoke 带有 SNS 目标的配置示例

在此示例中，为在 OnSuccess 配置的模板文件中声明的 SNS 主题提供了一个目标。

YAML

```
EventInvokeConfig:
  DestinationConfig:
    OnSuccess:
      Type: SNS
```

```
      Destination:
        Ref: DestinationSNS       # Arn of an SNS topic declared in the tempate file
```

EventSource

描述触发函数的事件源的对象。每个事件都由一个类型和一组依赖于该类型的属性组成。有关每个事件源的属性的更多信息，请参阅与具体类型对应的主题。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
 Properties: AlexaSkill | Api | CloudWatchEvent | CloudWatchLogs | Cognito
| DocumentDB | DynamoDB | EventBridgeRule | HttpApi | IoTRule | Kinesis | MQ | MSK
| S3 | Schedule | ScheduleV2 | SelfManagedKafka | SNS | SQS
 Type: String
```

属性

`Properties`

描述此事件映射的属性的对象。这组属性必须符合定义的类型。

类型 ：AlexaSkill| Ap i | | | Co gnito CloudWatchEventCloudWatchLogs| documentDB | D ynamo DB | | | Io | Kinesis EventBridgeRule| MQ HttpApi| MS K TRule | S3 | 日程安排 | Schedu leV2 | |SNS | SQ S SelfManagedKafka

必需：是

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`Type`

事件类型。

有效值：`AlexaSkill`, `Api`, `CloudWatchEvent`, `CloudWatchLogs`, `Cognito`, `DocumentDB`, `DynamoDB`, `EventBridgeRule`, `HttpApi`, `IoTRule`, `Kinesis`, `MQ`, `MSK`, `S3`, `Schedule`, `ScheduleV2`, `SelfManagedKafka`, `SNS`, `SQS`

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

示例

APIEvent

使用 API 事件的示例

YAML

```
ApiEvent:
  Type: Api
  Properties:
    Method: get
    Path: /group/{user}
    RestApiId:
      Ref: MyApi
```

AlexaSkill

描述 `AlexaSkill` 事件源类型的对象。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
  SkillId: String
```

属性

`SkillId`

Alexa 技能的 ID。有关技能 ID 的更多信息，请参阅 Alexa Skills Kit 文档中的为 Lambda 函数配置触发器。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

示例

AlexaSkillTrigger

Alexa Skill 事件示例

YAML

```
AlexaSkillEvent:
  Type: AlexaSkill
```

Api

描述 `Api` 事件源类型的对象。如果定义了 AWS::Serverless::Api 资源，则路径和方法值必须与 API 的 OpenAPI 定义中的操作相对应。

如果未定义 AWS::Serverless::Api，则函数的输入和输出表示 HTTP 请求和 HTTP 响应。

例如，使用 JavaScript API，可以通过返回带有 statusCode 和 body 密钥的对象来控制响应的状态代码和正文。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
  Auth: ApiFunctionAuth
  Method: String
  Path: String
  RequestModel: RequestModel
  RequestParameters: List of [ String | RequestParameter ]
  RestApiId: String
  TimeoutInMillis: Integer
```

属性

`Auth`

此特定 Api+路径+方法的身份验证配置。

对于未指定 `DefaultAuthorizer` 时覆盖在单个路径上 API 的 `DefaultAuthorizer` 设置身份验证配置或覆盖默认 `ApiKeyRequired` 设置很有用。

类型：ApiFunctionAuth

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`Method`

调用此函数的 HTTP 方法。选项包括`DELETE`、`GET`、`HEAD`、`OPTIONS`、`PATCH`、`POST`、`PUT`、和`ANY`。有关详细信息，请参阅《API Gateway 开发者指南》中的设置 HTTP 方法。

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`Path`

调用此函数的 URI 路径。必须以 / 开头。

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`RequestModel`

用于此特定 Api+路径+方法 的请求模型。这应该引用 AWS::Serverless::Api 资源 `Models` 部分中指定的模型的名称。

类型：RequestModel

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`RequestParameters`

此特定 Api+Path+Method 的请求参数配置。所有参数名称必须以 `method.request` 开头，且必须限制为 `method.request.header`、`method.request.querystring`、或`method.request.path`。

列表既可以包含参数名称字符串，也可以包含RequestParameter对象。对于字符串，`Required` 和 `Caching` 属性将默认为 `false`。

类型：[字符串 | RequestParameter] 列表

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`RestApiId`

RestApi 资源的标识符，必须包含具有给定路径和方法的操作。通常，将其设置为引用此模板中定义的 AWS::Serverless::Api 资源。

如果未定义此属性，则使用生成的`OpenApi`文档 Amazon SAM 创建默认AWS::Serverless::Api资源。该资源包含所有路径和方法的并集，这些路径和方法由同一模板中的 `Api` 事件定义，但未指定 `RestApiId`。

这不能引用其他模板中定义的 AWS::Serverless::Api 资源。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`TimeoutInMillis`

自定义超时值，范围在 50 到 29,000 毫秒之间。

> **Note**
>
> 当您指定此属性时， Amazon SAM 会修改您的 OpenAPI 定义。必须使用 `DefinitionBody` 属性内联指定 OpenAPI 定义。

类型：整数

必需：否

默认值：29,000 毫秒或 29 秒

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

示例

基本示例

YAML

```
Events:
  ApiEvent:
    Type: Api
    Properties:
      Path: /path
      Method: get
      RequestParameters:
        - method.request.header.Authorization
        - method.request.querystring.keyword:
            Required: true
            Caching: false
```

ApiFunctionAuth

在事件级别为特定 API、路径和方法配置授权。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
  ApiKeyRequired: Boolean
  AuthorizationScopes: List
  Authorizer: String
  InvokeRole: String
```

```
  OverrideApiAuth: Boolean
  ResourcePolicy: ResourcePolicyStatement
```

属性

`ApiKeyRequired`

此 API、路径和方法需要 API 密钥。

类型：布尔值

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`AuthorizationScopes`

适用于此 API、路径和方法的授权范围。

如果您已指定 `DefaultAuthorizer` 属性，则您指定的范围将覆盖该属性应用的所有范围。

类型：列表

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`Authorizer`

特定函数的`Authorizer`。

如果您为 `AWS::Serverless::Api` 资源指定了全局授权方，则可以通过将 `Authorizer` 设置为 `NONE` 来覆盖该授权方。有关示例，请参阅覆盖 Amazon API Gateway REST API 的全局授权方。

> **Note**
>
> 如果您使用 `AWS::Serverless::Api` 资源的 `DefinitionBody` 属性来描述您的 API，则必须使用 `OverrideApiAuth` 和 `Authorizer` 来覆盖您的全局授权方。请参阅`OverrideApiAuth`了解更多信息。

有效值：`AWS_IAMNONE`、或 Amazon SAM 模板中定义的任何授权者的逻辑 ID。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`InvokeRole`

指定用于 `AWS_IAM` 授权的 `InvokeRole`。

类型：字符串

必需：否

默认值：`CALLER_CREDENTIALS`

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

附加说明：`CALLER_CREDENTIALS` 映射到 `arn:aws:iam::`*`:<user>/`*，后者使用调用者凭证来调用端点。

`OverrideApiAuth`

指定为 `true` 以覆盖 `AWS::Serverless::Api` 资源的全局授权方配置。只有在您指定全局授权方并使用 `AWS::Serverless::Api` 资源的 `DefinitionBody` 属性来描述您的 API 时，才需要此属性。

> **Note**
>
> 当你指定`OverrideApiAuth`为时`true`， Amazon SAM 将使用为、或`ResourcePolicy`提供的任何值覆盖你的全局授权`ApiKeyRequired`者`Authorizer`。因此，使用 `OverrideApiAuth` 时还必须指定其中至少一个属性。有关示例，请参阅 在指定 for 时 DefinitionBody 覆盖全局授权 AWS::Serverless::Api 方。

类型：布尔值

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`ResourcePolicy`

在 API 上为此路径配置资源策略。

类型：ResourcePolicyStatement

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

示例

函数身份验证

以下示例指定了函数级别的授权。

YAML

```
Auth:
  ApiKeyRequired: true
  Authorizer: NONE
```

覆盖 Amazon API Gateway REST API 的全局授权方

您可以为 `AWS::Serverless::Api` 资源指定全局授权方。以下是配置全局默认授权方的示例：

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
...
Resources:
  MyApiWithLambdaRequestAuth:
    Type: AWS::Serverless::Api
    Properties:
      ...
      Auth:
        Authorizers:
          MyLambdaRequestAuth:
            FunctionArn: !GetAtt MyAuthFn.Arn
        DefaultAuthorizer: MyLambdaRequestAuth
```

要覆盖 Amazon Lambda 函数的默认授权者，可以指定`Authorizer`为`NONE`。以下是示例：

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
...
Resources:
  ...
  MyFn:
    Type: AWS::Serverless::Function
    Properties:
      ...
      Events:
        LambdaRequest:
          Type: Api
          Properties:
            RestApiId: !Ref MyApiWithLambdaRequestAuth
            Method: GET
            Auth:
              Authorizer: NONE
```

在指定 for 时 DefinitionBody 覆盖全局授权 AWS::Serverless::Api 方

当使用 `DefinitionBody` 属性描述 `AWS::Serverless::Api` 资源时，之前的覆盖方法不起作用。以下是对 `AWS::Serverless::Api` 资源使用 `DefinitionBody` 属性的示例：

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
...
Resources:
  MyApiWithLambdaRequestAuth:
    Type: AWS::Serverless::Api
    Properties:
      ...
      DefinitionBody:
        swagger: 2.0
        ...
        paths:
          /lambda-request:
            ...
      Auth:
        Authorizers:
          MyLambdaRequestAuth:
            FunctionArn: !GetAtt MyAuthFn.Arn
```

```
        DefaultAuthorizer: MyLambdaRequestAuth
```

要覆盖全局授权方，请使用 OverrideApiAuth 属性。以下是使用 OverrideApiAuth 通过为 Authorizer 提供的值覆盖全局授权方的示例：

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
...
Resources:
  MyApiWithLambdaRequestAuth:
    Type: AWS::Serverless::Api
    Properties:
      ...
      DefinitionBody:
        swagger: 2-0
        ...
        paths:
          /lambda-request:
            ...
      Auth:
        Authorizers:
          MyLambdaRequestAuth:
            FunctionArn: !GetAtt MyAuthFn.Arn
        DefaultAuthorizer: MyLambdaRequestAuth

    MyAuthFn:
      Type: AWS::Serverless::Function
      ...

    MyFn:
      Type: AWS::Serverless::Function
        Properties:
          ...
          Events:
            LambdaRequest:
              Type: Api
              Properties:
                RestApiId: !Ref MyApiWithLambdaRequestAuth
                Method: GET
                Auth:
                  Authorizer: NONE
                  OverrideApiAuth: true
                Path: /lambda-token
```

ResourcePolicyStatement

为 API 的所有方法和路径配置资源策略。有关资源策略的更多信息，请参阅《API Gateway 开发人员指南》中的使用 API Gateway 资源策略控制 API 的访问权限。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
AwsAccountBlacklist: List
AwsAccountWhitelist: List
CustomStatements: List
IntrinsicVpcBlacklist: List
IntrinsicVpcWhitelist: List
IntrinsicVpceBlacklist: List
IntrinsicVpceWhitelist: List
IpRangeBlacklist: List
IpRangeWhitelist: List
SourceVpcBlacklist: List
SourceVpcWhitelist: List
```

属性

`AwsAccountBlacklist`

要封锁的 Amazon 账户。

类型：字符串列表

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`AwsAccountWhitelist`

要允许的 Amazon 账户。有关此属性的使用示例，请参阅本页底部的“示例”部分。

类型：字符串列表

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`CustomStatements`

适用于此 API 的自定义资源策略语句列表。有关此属性的使用示例，请参阅本页底部的“示例”部分。

类型：列表

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`IntrinsicVpcBlacklist`

要屏蔽的虚拟私有云列表 (VPCs)，其中每个 VPC 都指定为引用，例如动态引用或`Ref`内部函数。有关此属性的使用示例，请参阅本页底部的“示例”部分。

类型：列表

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`IntrinsicVpcWhitelist`

VPCs 要允许的列表，其中每个 VPC 都指定为引用，例如动态引用或`Ref`内部函数。

类型：列表

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`IntrinsicVpceBlacklist`

要阻止的 VPC 端点列表，其中每个 VPC 端点都指定为引用，例如动态引用或`Ref`内置函数。

类型：列表

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`IntrinsicVpceWhitelist`

要允许的 VPC 端点列表，其中每个 VPC 端点都指定为引用，例如动态引用或`Ref`内置函数。有关此属性的使用示例，请参阅本页底部的“示例”部分。

类型：列表

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`IpRangeBlacklist`

要阻止的 IP 地址或地址范围。有关此属性的使用示例，请参阅本页底部的“示例”部分。

类型：列表

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`IpRangeWhitelist`

要允许的 IP 地址或地址范围。

类型：列表

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`SourceVpcBlacklist`

要阻止的源 VPC 或 VPC 端点。源 VPC 名称必须以 `"vpc-"` 开头，源 VPC 端点名称必须以 `"vpce-"` 开头。有关此属性的使用示例，请参阅本页底部的“示例”部分。

类型：列表

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`SourceVpcWhitelist`

要允许的源 VPC 或 VPC 端点。源 VPC 名称必须以 `"vpc-"` 开头，源 VPC 端点名称必须以 `"vpce-"` 开头。

类型：列表

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

示例

资源策略示例

以下示例屏蔽两个 IP 地址和一个源 VPC，并允许一个 Amazon 账户。

YAML

```
Auth:
  ResourcePolicy:
    CustomStatements: [{
                        "Effect": "Allow",
                        "Principal": "*",
                        "Action": "execute-api:Invoke",
                        "Resource": "execute-api:/Prod/GET/pets",
                        "Condition": {
                          "IpAddress": {
                            "aws:SourceIp": "1.2.3.4"
                          }
                        }
                      }]
    IpRangeBlacklist:
      - "10.20.30.40"
      - "1.2.3.4"
    SourceVpcBlacklist:
      - "vpce-1a2b3c4d"
    AwsAccountWhitelist:
      - "111122223333"
```

```
    IntrinsicVpcBlacklist:
      - "{{resolve:ssm:SomeVPCReference:1}}"
      - !Ref MyVPC
    IntrinsicVpceWhitelist:
      - "{{resolve:ssm:SomeVPCEReference:1}}"
      - !Ref MyVPCE
```

RequestModel

为特定 Api+Path+ 方法配置请求模型。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
Model: String
Required: Boolean
ValidateBody: Boolean
ValidateParameters: Boolean
```

属性

`Model`

在 AWS::Serverless::Api Models 属性中定义的模型的名称。

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`Required`

在给定 API 端点的 OpenApi 定义的参数部分中添加一个`required`属性。

类型：布尔值

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`ValidateBody`

指定 API Gateway 是否使用 `Model` 验证请求正文。有关更多信息，请参阅《API Gateway 开发人员指南》中的在 API Gateway 中启用请求验证。

类型：布尔值

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`ValidateParameters`

指定 API Gateway 是否使用 `Model` 验证请求路径参数、查询字符串和标头。有关更多信息，请参阅《API Gateway 开发人员指南》中的在 API Gateway 中启用请求验证。

类型：布尔值

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

示例

请求模型

请求模型示例

YAML

```
RequestModel:
  Model: User
  Required: true
  ValidateBody: true
  ValidateParameters: true
```

RequestParameter

为特定 Api+路径+方法配置请求参数。

需要为请求参数指定 `Required` 或 `Caching` 属性

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
Caching: Boolean
Required: Boolean
```

属性

`Caching`

在 API Gateway OpenApi 定义中添加`cacheKeyParameters`章节

类型：布尔值

必需：条件

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`Required`

此字段指定是否需要参数。

类型：布尔值

必需：条件

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

示例

请求参数

设置请求参数的示例

YAML

```
RequestParameters:
```

```
  - method.request.header.Authorization:
      Required: true
      Caching: true
```

CloudWatchEvent

描述 `CloudWatchEvent` 事件源类型的对象。

Amazon Serverless Application Model (Amazon SAM) 生成一个 AWS::Events::Rule设置此事件类型时的资源。

重要说明：EventBridgeRule是首选使用的事件源类型，而不是`CloudWatchEvent`。`EventBridgeRule`并`CloudWatchEvent`使用相同的底层服务、API 和 Amazon CloudFormation 资源。但是， Amazon SAM 将仅向添加对新功能的支持`EventBridgeRule`。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
Enabled: Boolean
EventBusName: String
Input: String
InputPath: String
Pattern: EventPattern
State: String
```

属性

`Enabled`

指示是否启用规则。

要禁用该规则，请将此属性设置为 `false`。

> **Note**
>
> 指定 `Enabled` 或 `State` 属性，但不能同时指定两者。

类型：布尔值

必需：否

Amazon CloudFormation 兼容性：此属性类似于`AWS::Events::Rule`资源的`State`属性。如果此属性设置为，`true`则 Amazon SAM 通过`ENABLED`，否则通过`DISABLED`。

`EventBusName`

要与该规则关联的事件总线。如果省略此属性，则 Amazon SAM 使用默认的事件总线。

类型：字符串

必需：否

默认：默认事件总线

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Events::Rule`资源的`EventBusName`属性。

`Input`

传递到目标的有效 JSON 文本。如果使用此属性，则不会将事件文本本身的任何内容传递到目标。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Events::Rule Target`资源的`Input`属性。

`InputPath`

当您不希望将整个匹配事件传递给目标时，请使用 `InputPath` 属性描述要传递事件的哪一部分。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Events::Rule Target`资源的`InputPath`属性。

`Pattern`

描述哪些事件路由到指定目标。有关更多信息，请参阅 Amazon EventBridge 用户指南 EventBridge中的事件和事件模式。

类型：EventPattern

必需：是

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Events::Rule`资源的`EventPattern`属性。

`State`

规则的状态。

接受的值：`DISABLED | ENABLED`

> **Note**
> 指定 `Enabled` 或 `State` 属性，但不能同时指定两者。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Events::Rule`资源的`State`属性。

示例

CloudWatchEvent

以下是 `CloudWatchEvent` 事件源类型的示例。

YAML

```
CWEvent:
  Type: CloudWatchEvent
  Properties:
    Enabled: false
    Input: '{"Key": "Value"}'
    Pattern:
      detail:
        state:
          - running
```

CloudWatchLogs

描述 `CloudWatchLogs` 事件源类型的对象。

此事件生成一个 AWS::Logs::SubscriptionFilterresource 并指定订阅筛选器并将其与指定的日志组相关联。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
FilterPattern: String
LogGroupName: String
```

属性

`FilterPattern`

限制传送到目标 Amazon 资源的内容的筛选表达式。有关筛选器模式语法的更多信息，请参阅筛选器和模式语法。

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Logs::SubscriptionFilter`资源的`FilterPattern`属性。

`LogGroupName`

要与订阅筛选器关联的日志组。如果筛选模式与日志事件匹配，则所有上传到该日志组的日志事件都将被筛选并传送到指定的 Amazon 资源。

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Logs::SubscriptionFilter`资源的`LogGroupName`属性。

示例

Cloudwatchlogs 订阅筛选器

Cloudwatchlog 订阅筛选示例

YAML

```
CWLog:
  Type: CloudWatchLogs
  Properties:
    LogGroupName:
      Ref: CloudWatchLambdaLogsGroup
    FilterPattern: My pattern
```

Cognito

描述 `Cognito` 事件源类型的对象。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
  Trigger: List
  UserPool: String
```

属性

`Trigger`

新用户群体的 Lambda 触发器配置信息。

类型：列表

必需：是

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Cognito::UserPool`资源的`LambdaConfig`属性。

`UserPool`

对在同一模板中 UserPool 定义的引用

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

示例

Cognito 事件

Cognito 事件示例

YAML

```
CognitoUserPoolPreSignup:
  Type: Cognito
  Properties:
    UserPool:
      Ref: MyCognitoUserPool
    Trigger: PreSignUp
```

DocumentDB

描述 `DocumentDB` 事件源类型的对象。有关更多信息，请参阅Amazon Lambda 开发者指南中的Amazon Lambda 与亚马逊 DocumentDB 配合使用。

语法

要在 Amazon SAM 模板中声明此实体，请使用以下语法。

YAML

```
BatchSize: Integer
Cluster: String
CollectionName: String
DatabaseName: String
Enabled: Boolean
FilterCriteria: FilterCriteria
FullDocument: String
KmsKeyArn: String
MaximumBatchingWindowInSeconds: Integer
```

```
SecretsManagerKmsKeyId: String
SourceAccessConfigurations: List
StartingPosition: String
StartingPositionTimestamp: Double
```

属性

BatchSize

要在单个批次中检索的最大项目数。

类型：整数

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::EventSourceMapping`资源的 `BatchSize`属性。

Cluster

Amazon DocumentDB 集群的 Amazon 资源名称（ARN）。

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::EventSourceMapping`资源的 `EventSourceArn`属性。

CollectionName

将使用的数据库中集合的名称。如果您未指定集合，Lambda 会使用所有集合。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::EventSourceMappingDocumentDBEventSourceConfig`数据类型的 `CollectionName`属性。

DatabaseName

将使用的 Amazon DocumentDB 集群中数据库的名称。

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::EventSourceMappingDocumentDBEventSourceConfig`数据类型的`DatabaseName`属性。

`Enabled`

如果为 `true`，则事件源映射处于活动状态。要暂停轮询和调用，设置为 `false`。

类型：布尔值

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::EventSourceMapping`资源的`Enabled`属性。

`FilterCriteria`

定义用于确定 Lambda 是否应处理事件的条件的对象。有关更多信息，请参阅《Amazon Lambda 开发人员指南》中的 Lambda 事件筛选。

类型：FilterCriteria

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::EventSourceMapping`资源的`FilterCriteria`属性。

`FullDocument`

确定 Amazon DocumentDB 在文档更新操作期间将向您的事件流发送的内容。如果设置为 `UpdateLookup`，Amazon DocumentDB 将发送一个描述所发生更改的增量以及完整文档的副本。否则，Amazon DocumentDB 将仅发送包含更改的部分文档。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::EventSourceMappingDocumentDBEventSourceConfig`数据类型的`FullDocument`属性。

`KmsKeyArn`

用于加密与此事件相关信息的密钥的 Amazon 资源名称（ARN）。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::EventSourceMapping`资源的`KmsKeyArn`属性。

`MaximumBatchingWindowInSeconds`

在调用函数之前收集记录的最长时间（以秒为单位）。

类型：整数

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::EventSourceMapping`资源的 `MaximumBatchingWindowInSeconds`属性。

`SecretsManagerKmsKeyId`

来自 Secrets Manager 的客户托管密钥的 Amazon Key Management Service (Amazon KMS) Amazon 密钥 ID。当您以不包含 `kms:Decrypt` 权限的 Lambda 执行角色使用来自 Secrets Manager 的客户托管密钥时，此为必需。

此属性的值为 UUID。例如：`1abc23d4-567f-8ab9-cde0-1fab234c5d67`。

类型：字符串

必需：条件

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`SourceAccessConfigurations`

身份验证协议或虚拟主机的数组。使用`SourceAccessConfigurations`数据类型进行指定。

对于 `DocumentDB` 事件源类型，唯一有效的配置类型是 `BASIC_AUTH`。

- `BASIC_AUTH` – 存储您的代理凭证的 Secrets Manager 密钥。要使用此类型，凭证必须采用以下格式：`{"username": "your-username", "password": "your-password"}`。只允许使用一个类型为 `BASIC_AUTH` 的对象。

类型：列表

必需：是

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::EventSourceMapping`资源的 `SourceAccessConfigurations`属性。

`StartingPosition`

在流中开始读取数据的位置。

- `AT_TIMESTAMP` – 指定开始读取记录的时间。
- `LATEST` - 仅读取新记录。
- `TRIM_HORIZON` - 处理所有可用的记录。

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::EventSourceMapping`资源的 `StartingPosition`属性。

`StartingPositionTimestamp`

开始读取的时间（以 Unix 时间秒为单位）在 `StartingPosition` 被指定为 `AT_TIMESTAMP` 的情况下定义 `StartingPositionTimestamp`。

类型：双精度

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::EventSourceMapping`资源的 `StartingPositionTimestamp`属性。

示例

Amazon DocumentDB 事件源

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
...
Resources:
```

```
MyFunction:
  Type: AWS::Serverless::Function
  Properties:
  ...
    Events:
      MyDDBEvent:
        Type: DocumentDB
        Properties:
          Cluster: "arn:aws:rds:us-west-2:123456789012:cluster:docdb-2023-01-01"
          BatchSize: 10
          MaximumBatchingWindowInSeconds: 5
          DatabaseName: "db1"
          CollectionName: "collection1"
          FullDocument: "UpdateLookup"
          SourceAccessConfigurations:
            - Type: BASIC_AUTH
              URI: "arn:aws:secretsmanager:us-west-2:123456789012:secret:doc-db"
```

DynamoDB

描述 DynamoDB 事件源类型的对象。有关更多信息，请参阅开发者指南中的Amazon LambdaAmazon Lambda 与亚马逊 DynamoDB 搭配使用。

Amazon SAM 生成一个 AWS::Lambda::EventSourceMapping设置此事件类型时的资源。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
BatchSize: Integer
BisectBatchOnFunctionError: Boolean
DestinationConfig: DestinationConfig
Enabled: Boolean
FilterCriteria: FilterCriteria
FunctionResponseTypes: List
KmsKeyArn: String
MaximumBatchingWindowInSeconds: Integer
MaximumRecordAgeInSeconds: Integer
MaximumRetryAttempts: Integer
MetricsConfig: MetricsConfig
ParallelizationFactor: Integer
```

```
  StartingPosition: String
  StartingPositionTimestamp: Double
  Stream: String
  TumblingWindowInSeconds: Integer
```

属性

`BatchSize`

要在单个批次中检索的最大项目数。

类型：整数

必需：否

默认值：100

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::EventSourceMapping`资源的`BatchSize`属性。

最小值：1

最大值：1000

`BisectBatchOnFunctionError`

如果函数返回错误，则将批次拆分为两批并重试。

类型：布尔值

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::EventSourceMapping`资源的`BisectBatchOnFunctionError`属性。

`DestinationConfig`

丢弃的记录的 Amazon Simple Queue Service (Amazon SQS) 队列或 Amazon Simple Notification Service (Amazon SNS)主题目标。

类型：DestinationConfig

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::EventSourceMapping`资源的`DestinationConfig`属性。

`Enabled`

禁用事件源映射以暂停轮询和调用。

类型：布尔值

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::EventSourceMapping`资源的`Enabled`属性。

`FilterCriteria`

定义用于确定 Lambda 是否应处理事件的条件的对象。有关更多信息，请参阅《Amazon Lambda 开发人员指南》中的 Amazon Lambda 事件筛选。

类型：FilterCriteria

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::EventSourceMapping`资源的`FilterCriteria`属性。

`FunctionResponseTypes`

当前应用于事件源映射的响应类型的列表。有关详细信息，请参阅《Amazon Lambda 开发人员指南》中的报告批处理项目失败。

有效值：`ReportBatchItemFailures`

类型：列表

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::EventSourceMapping`资源的`FunctionResponseTypes`属性。

`KmsKeyArn`

用于加密与此事件相关信息的密钥的 Amazon 资源名称（ARN）。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::EventSourceMapping`资源的`KmsKeyArn`属性。

`MaximumBatchingWindowInSeconds`

在调用函数之前收集记录的最长时间（以秒为单位）。

类型：整数

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::EventSourceMapping`资源的`MaximumBatchingWindowInSeconds`属性。

`MaximumRecordAgeInSeconds`

Lambda 发送到函数以进行处理的记录的最长期限。

类型：整数

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::EventSourceMapping`资源的`MaximumRecordAgeInSeconds`属性。

`MaximumRetryAttempts`

在函数返回错误时重试的最大次数。

类型：整数

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::EventSourceMapping`资源的`MaximumRetryAttempts`属性。

`MetricsConfig`

一种可选配置，用于获取捕获每个处理阶段的事件源映射的增强指标。有关示例，请参阅MetricsConfig 事件。

类型：MetricsConfig

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给AWS::Lambda::EventSourceMapping资源的MetricsConfig属性。

ParallelizationFactor

要从每个分片中同时处理的批次数。

类型：整数

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给AWS::Lambda::EventSourceMapping资源的ParallelizationFactor属性。

StartingPosition

在流中开始读取数据的位置。

- AT_TIMESTAMP – 指定开始读取记录的时间。
- LATEST - 仅读取新记录。
- TRIM_HORIZON - 处理所有可用的记录。

有效值：AT_TIMESTAMP | LATEST | TRIM_HORIZON

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性直接传递给AWS::Lambda::EventSourceMapping资源的StartingPosition属性。

StartingPositionTimestamp

开始读取的时间（以 Unix 时间秒为单位）在 StartingPosition 被指定为 AT_TIMESTAMP 的情况下定义 StartingPositionTimestamp。

类型：双精度

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给AWS::Lambda::EventSourceMapping资源的StartingPositionTimestamp属性。

`Stream`

DynamoDB 流的 Amazon 资源名称（ARN）。

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::EventSourceMapping`资源的`EventSourceArn`属性。

`TumblingWindowInSeconds`

处理窗口的持续时间（以秒为单位）。有效范围为 1 到 900（15 分钟）。

有关更多信息，请参阅《Amazon Lambda 开发人员指南》的滚动窗口。

类型：整数

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::EventSourceMapping`资源的`TumblingWindowInSeconds`属性。

示例

MetricsConfig 事件

以下是一个资源示例，该资源使用该`MetricsConfig`属性捕获其事件源映射的每个处理阶段。

```
Resources:
  FilteredEventsFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: s3://sam-demo-bucket/metricsConfig.zip
      Handler: index.handler
      Runtime: nodejs16.x
      Events:
        KinesisStream:
          Type: Kinesis
          Properties:
            Stream: !GetAtt KinesisStream.Arn
            StartingPosition: LATEST
            MetricsConfig:
```

```
                Metrics:
                - EventCount
```

现有 DynamoDB 表的 DynamoDB 事件源

账户中已存在的 DynamoDB 表的 DynamoDB 事件源。 Amazon

YAML

```
Events:
  DDBEvent:
    Type: DynamoDB
    Properties:
      Stream: arn:aws:dynamodb:us-east-1:123456789012:table/TestTable/
stream/2016-08-11T21:21:33.291
      StartingPosition: TRIM_HORIZON
      BatchSize: 10
      Enabled: false
```

模板中声明的 DynamoDB 表的 DynamoDB 事件

在同一模板文件中声明的 DynamoDB 表的 DynamoDB 事件。

YAML

```
Events:
  DDBEvent:
    Type: DynamoDB
    Properties:
      Stream:
        !GetAtt MyDynamoDBTable.StreamArn   # This must be the name of a DynamoDB table
 declared in the same template file
      StartingPosition: TRIM_HORIZON
      BatchSize: 10
      Enabled: false
```

EventBridgeRule

描述`EventBridgeRule`事件源类型的对象，它将您的无服务器函数设置为 Amazon EventBridge 规则的目标。有关更多信息，请参阅什么是亚马逊 EventBridge？ 在《亚马逊 EventBridge 用户指南》中。

Amazon SAM 生成一个 AWS::Events::Rule设置此事件类型时的资源。 Amazon SAM 还会创建`AWS::Lambda::Permission`资源，该资源是调用 Lambda 所`EventBridgeRule`必需的。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
DeadLetterConfig: DeadLetterConfig
EventBusName: String
Input: String
InputPath: String
InputTransformer: InputTransformer
Pattern: EventPattern
RetryPolicy: RetryPolicy
RuleName: String
State: String
Target: Target
```

属性

`DeadLetterConfig`

配置亚马逊简单队列服务 (Amazon SQS) Simple Queue Service 队列，目标调用失败后 EventBridge 在该队列中发送事件。例如，当向不存在的 Lambda 函数发送事件时，或者没有足够的权限调用 Lambda 函数 EventBridge 时，调用可能会失败。有关更多信息，请参阅 A ma EventBridge zon 用户指南中的事件重试策略和使用死信队列。

> **Note**
>
> AWS::Serverless::Function 资源类型具有类似的数据类型 `DeadLetterQueue`，用于处理成功调用目标 Lambda 函数后发生的故障。这些类型的故障示例包括 Lambda 节流或 Lambda 目标函数返回的错误。有关函数 `DeadLetterQueue` 属性的更多信息，请参阅《Amazon Lambda 开发人员指南》中的死信队列。

类型：DeadLetterConfig

必需：否

Amazon CloudFormation 兼容性：此属性类似于`AWS::Events::RuleTarget`数据类型的`DeadLetterConfig`属性。此属性的 Amazon SAM 版本包括其他子属性， Amazon SAM 以备您想要创建死信队列时使用。

`EventBusName`

要与该规则关联的事件总线。如果省略此属性，则 Amazon SAM 使用默认的事件总线。

类型：字符串

必需：否

默认：默认事件总线

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Events::Rule`资源的`EventBusName`属性。

`Input`

传递到目标的有效 JSON 文本。如果使用此属性，则不会将事件文本本身的任何内容传递到目标。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Events::Rule Target`资源的`Input`属性。

`InputPath`

当您不希望将整个匹配事件传递给目标时，请使用 `InputPath` 属性描述要传递事件的哪一部分。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Events::Rule Target`资源的`InputPath`属性。

`InputTransformer`

使您可以根据特定事件数据向目标提供自定义输入的设置。您可以从事件中提取一个或多个键值对，然后使用该数据将自定义输入发送到目标。有关更多信息，请参阅《亚马逊 EventBridge 用户指南》中的亚马逊 EventBridge 输入转换。

类型：InputTransformer

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Events::RuleTarget`数据类型的`InputTransformer`属性。

`Pattern`

描述哪些事件路由到指定目标。有关更多信息，请参阅《亚马逊 EventBridge 用户指南》中的亚马逊 EventBridgeEventBridge 事件和事件模式。

类型：EventPattern

必需：是

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Events::Rule`资源的`EventPattern`属性。

`RetryPolicy`

包含有关重试策略设置的信息的 `RetryPolicy` 对象。有关更多信息，请参阅 A ma EventBridge zon 用户指南中的事件重试策略和使用死信队列。

类型：RetryPolicy

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Events::RuleTarget`数据类型的`RetryPolicy`属性。

`RuleName`

规则的名称。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Events::Rule`资源的`Name`属性。

`State`

规则的状态。

可接受的值：`DISABLED`| `ENABLED` | `ENABLED_WITH_ALL_CLOUDTRAIL_MANAGEMENT_EVENTS`

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Events::Rule`资源的`State` 属性。

`Target`

触发规则时 EventBridge 调用的 Amazon 资源。您可以使用此属性来指定目标的逻辑 ID。如果未指定此属性，则 Amazon SAM 生成目标的逻辑 ID。

类型：目标

必需：否

Amazon CloudFormation 兼容性：此属性类似于`AWS::Events::Rule`资源的`Targets`属性。`Amazon EC2 RebootInstances API call`是目标属性的一个示例。此属性的 Amazon SAM 版本仅允许您指定单个目标的逻辑 ID。

示例

EventBridgeRule

以下是 `EventBridgeRule` 事件源类型的示例。

YAML

```
EBRule:
  Type: EventBridgeRule
  Properties:
    Input: '{"Key": "Value"}'
    Pattern:
      detail:
        state:
          - terminated
    RetryPolicy:
      MaximumRetryAttempts: 5
      MaximumEventAgeInSeconds: 900
    DeadLetterConfig:
      Type: SQS
      QueueLogicalId: EBRuleDLQ
    Target:
      Id: MyTarget
```

DeadLetterConfig

用于指定亚马逊简单队列服务 (Amazon SQS) Simple Queue Service 队列的对象，目标调用失败后 EventBridge 在该队列中发送事件。例如，在向不存在的 Lambda 函数发送事件或调用 Lambda 函数的权限不足时，调用可能会失败。有关更多信息，请参阅 A ma EventBridge zon 用户指南中的事件重试策略和使用死信队列。

> **Note**
>
> AWS::Serverless::Function 资源类型具有类似的数据类型 `DeadLetterQueue`，用于处理成功调用目标 Lambda 函数后发生的故障。此类故障的示例包括 Lambda 节流或 Lambda 目标函数返回的错误。有关函数 `DeadLetterQueue` 属性的更多信息，请参阅《Amazon Lambda 开发人员指南》中的死信队列。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```yaml
Arn: String
QueueLogicalId: String
Type: String
```

属性

`Arn`

指定作为死信队列目标的 Amazon SQS 队列的 Amazon 资源名称（ARN）。

> **Note**
>
> 指定 `Type` 属性或 `Arn` 属性，但不能同时指定两者。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Events::RuleDeadLetterConfig`数据类型的`Arn`属性。

`QueueLogicalId`

指定了 Amazon SAM 创建的死信队列的自定义名称。`Type`

> **Note**
>
> 如果未设置 `Type` 属性，则将忽略该属性。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`Type`

队列的类型。设置此属性后， Amazon SAM 会自动创建死信队列并附加必要的基于资源的策略，以授予规则资源向队列发送事件的权限。

> **Note**
>
> 指定 `Type` 属性或 `Arn` 属性，但不能同时指定两者。

有效值：`SQS`

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

示例

DeadLetterConfig

DeadLetterConfig

YAML

```
DeadLetterConfig:
  Type: SQS
  QueueLogicalId: MyDLQ
```

Target

配置触发规则时 EventBridge 调用的 Amazon 资源。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
  Id: String
```

属性

`Id`

目标的逻辑 ID。

`Id` 的值可以包含字母数字字符、句点 (`.`)、连字符 (`-`) 和下划线 (`_`)。

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Events::RuleTarget`数据类型的`Id`属性。

示例

目标

YAML

```
EBRule:
  Type: EventBridgeRule
```

```
  Properties:
    Target:
      Id: MyTarget
```

HttpApi

描述带有类型的事件源的对象 HttpApi。

如果 API 上存在指定路径和方法的 OpenApi 定义，SAM 将为您添加 Lambda 集成和安全部分（如果适用）。

如果 API 上没有指定路径和方法的 OpenApi 定义，SAM 将为您创建此定义。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
ApiId: String
Auth: HttpApiFunctionAuth
Method: String
Path: String
PayloadFormatVersion: String
RouteSettings: RouteSettings
TimeoutInMillis: Integer
```

属性

`ApiId`

此模板中定义的 AWS::Serverless::HttpApi 资源的标识符。

如果未定义，则`ServerlessHttpApi`使用生成的 OpenApi 文档创建默认AWS::Serverless::HttpApi资源，该文档包含由此模板中定义的 Api 事件定义的所有路径和方法的并集，这些路径和方法未指定`ApiId`。

这不能引用其他模板中定义的 AWS::Serverless::HttpApi 资源。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`Auth`

此特定 Api+路径+方法的身份验证配置。

在未指定 `DefaultAuthorizer` 的情况下，对于覆盖 API 的 `DefaultAuthorizer` 或在各个路径中设置身份验证配置很有用。

类型：HttpApiFunctionAuth

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`Method`

调用此函数的 HTTP 方法。

如果未指定 `Path` 和 `Method`，SAM 会创建默认 API 路径，用于将未映射到其他端点的任何请求路由到此 Lambda 函数。每个 API 只能存在其中一个默认路径。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`Path`

调用此函数的 URI 路径。必须以 / 开头。

如果未指定 `Path` 和 `Method`，SAM 会创建默认 API 路径，用于将未映射到其他端点的任何请求路由到此 Lambda 函数。每个 API 只能存在其中一个默认路径。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`PayloadFormatVersion`

指定发送到集成的有效负载的格式。

注意： PayloadFormatVersion 需要 SAM 修改您的 OpenAPI 定义，因此它仅适用于属性中 OpenApi 定义的内联定义。`DefinitionBody`

类型：字符串

必需：否

默认值：2.0

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`RouteSettings`

此 HTTP API 的每条路径的路由设置。有关路由设置的更多信息，请参阅 API Gateway 开发者指南AWS::ApiGatewayV2::Stage RouteSettings中的。

注意：如果 RouteSettings 在 HttpApi 资源和事件源中同时指定，则将其 Amazon SAM 合并，优先使用事件源属性。

类型：RouteSettings

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::ApiGatewayV2::Stage`资源的`RouteSettings`属性。

`TimeoutInMillis`

自定义超时值，范围在 50 到 29,000 毫秒之间。

注意： TimeoutInMillis 需要 SAM 修改您的 OpenAPI 定义，因此它仅适用于属性中 OpenApi 定义的内联定义。`DefinitionBody`

类型：整数

必需：否

默认值：5000

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

示例

默认 HttpApi 事件

HttpApi 使用默认路径的事件。此 API 中所有未映射的路径和方法都将路由到此端点。

YAML

```
Events:
  HttpApiEvent:
    Type: HttpApi
```

HttpApi

HttpApi 使用特定路径和方法的事件。

YAML

```
Events:
  HttpApiEvent:
    Type: HttpApi
    Properties:
      Path: /
      Method: GET
```

HttpApi 授权

HttpApi 使用授权者的事件。

YAML

```
Events:
  HttpApiEvent:
    Type: HttpApi
    Properties:
      Path: /authenticated
      Method: GET
      Auth:
        Authorizer: OpenIdAuth
```

```
        AuthorizationScopes:
          - scope1
          - scope2
```

HttpApiFunctionAuth

在事件级别配置授权。

为特定 API+路径+方法配置身份验证

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
AuthorizationScopes: List
Authorizer: String
```

属性

`AuthorizationScopes`

适用于此 API、路径和方法的授权范围。

此处列出的范围将覆盖`DefaultAuthorizer`应用的任何范围（如果存在）。

类型：列表

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`Authorizer`

特定函数的`Authorizer`。要使用 IAM 授权，请在模板的 `Globals` 部分指定 `AWS_IAM`，并为 `EnableIamAuthorizer` 指定 `true`。

如果您已在 API 中指定了全局授权方并想公开特定函数，请通过将 `Authorizer` 设置为 `NONE` 来覆盖。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

示例

函数身份验证

在函数级别指定授权

YAML

```
Auth:
  Authorizer: OpenIdAuth
  AuthorizationScopes:
    - scope1
    - scope2
```

IAM 授权

在事件级别指定 IAM 授权。要在事件级别使用 `AWS_IAM` 授权，还必须在模板的 `Globals` 部分为 `EnableIamAuthorizer` 指定 `true`。有关更多信息，请参阅 Amazon SAM 模板的全局变量部分。

YAML

```
Globals:
  HttpApi:
    Auth:
      EnableIamAuthorizer: true

Resources:
  HttpApiFunctionWithIamAuth:
    Type: AWS::Serverless::Function
    Properties:
      Events:
        ApiEvent:
          Type: HttpApi
          Properties:
            Path: /iam-auth
            Method: GET
            Auth:
```

```
            Authorizer: AWS_IAM
      Handler: index.handler
      InlineCode: |
        def handler(event, context):
          return {'body': 'HttpApiFunctionWithIamAuth', 'statusCode': 200}
      Runtime: python3.9
```

IoTRule

描述 `IoTRule` 事件源类型的对象。

创建一个 AWS::IoT::TopicRule用于声明 Amazon IoT 规则的资源。有关更多信息，请参阅 Amazon CloudFormation 文档

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
AwsIotSqlVersion: String
Sql: String
```

属性

`AwsIotSqlVersion`

评估规则时使用的 SQL 规则引擎的版本。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::IoT::TopicRule TopicRulePayload`资源的`AwsIotSqlVersion`属性。

`Sql`

用于查询主题的 SQL 语句。有关更多信息，请参阅《Amazon IoT 开发人员指南》中的Amazon IoT SQL 参考。

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::IoT::TopicRule` `TopicRulePayload`资源的`Sql`属性。

示例

IoT 规则

IoT 规则示例

YAML

```
IoTRule:
  Type: IoTRule
  Properties:
    Sql: SELECT * FROM 'topic/test'
```

Kinesis

描述 `Kinesis` 事件源类型的对象。有关更多信息，请参阅《Amazon Lambda 开发者指南》中的Amazon Lambda 与 Amazon Kinesis 配合使用。

Amazon SAM 生成一个 AWS::Lambda::EventSourceMapping设置此事件类型时的资源。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
  BatchSize: Integer
  BisectBatchOnFunctionError: Boolean
  DestinationConfig: DestinationConfig
  Enabled: Boolean
  FilterCriteria: FilterCriteria
  FunctionResponseTypes: List
  KmsKeyArn: String
  MaximumBatchingWindowInSeconds: Integer
  MaximumRecordAgeInSeconds: Integer
  MaximumRetryAttempts: Integer
  MetricsConfig: MetricsConfig
```

```
  ParallelizationFactor: Integer
  StartingPosition: String
  StartingPositionTimestamp: Double
  Stream: String
  TumblingWindowInSeconds: Integer
```

属性

`BatchSize`

要在单个批次中检索的最大项目数。

类型：整数

必需：否

默认值：100

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::EventSourceMapping`资源的`BatchSize`属性。

最小值：1

最大值：10000

`BisectBatchOnFunctionError`

如果函数返回错误，则将批次拆分为两批并重试。

类型：布尔值

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::EventSourceMapping`资源的`BisectBatchOnFunctionError`属性。

`DestinationConfig`

丢弃的记录的 Amazon Simple Queue Service (Amazon SQS) 队列或 Amazon Simple Notification Service (Amazon SNS)主题目标。

类型：DestinationConfig

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::EventSourceMapping`资源的`DestinationConfig`属性。

`Enabled`

禁用事件源映射以暂停轮询和调用。

类型：布尔值

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::EventSourceMapping`资源的`Enabled`属性。

`FilterCriteria`

定义用于确定 Lambda 是否应处理事件的条件的对象。有关更多信息，请参阅《Amazon Lambda 开发人员指南》中的 Amazon Lambda 事件筛选。

类型：FilterCriteria

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::EventSourceMapping`资源的`FilterCriteria`属性。

`FunctionResponseTypes`

当前应用于事件源映射的响应类型的列表。有关详细信息，请参阅《Amazon Lambda 开发人员指南》中的报告批处理项目失败。

有效值：`ReportBatchItemFailures`

类型：列表

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::EventSourceMapping`资源的`FunctionResponseTypes`属性。

`KmsKeyArn`

用于加密与此事件相关信息的密钥的 Amazon 资源名称（ARN）。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::EventSourceMapping`资源的`KmsKeyArn`属性。

`MaximumBatchingWindowInSeconds`

在调用函数之前收集记录的最长时间（以秒为单位）。

类型：整数

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::EventSourceMapping`资源的`MaximumBatchingWindowInSeconds`属性。

`MaximumRecordAgeInSeconds`

Lambda 发送到函数以进行处理的记录的最长期限。

类型：整数

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::EventSourceMapping`资源的`MaximumRecordAgeInSeconds`属性。

`MaximumRetryAttempts`

在函数返回错误时重试的最大次数。

类型：整数

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::EventSourceMapping`资源的`MaximumRetryAttempts`属性。

`MetricsConfig`

一种可选配置，用于获取捕获每个处理阶段的事件源映射的增强指标。有关示例，请参阅MetricsConfig 事件。

类型：MetricsConfig

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给AWS::Lambda::EventSourceMapping资源的MetricsConfig属性。

ParallelizationFactor

要从每个分片中同时处理的批次数。

类型：整数

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给AWS::Lambda::EventSourceMapping资源的ParallelizationFactor属性。

StartingPosition

在流中开始读取数据的位置。

- AT_TIMESTAMP – 指定开始读取记录的时间。
- LATEST - 仅读取新记录。
- TRIM_HORIZON - 处理所有可用的记录。

有效值：AT_TIMESTAMP | LATEST | TRIM_HORIZON

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性直接传递给AWS::Lambda::EventSourceMapping资源的StartingPosition属性。

StartingPositionTimestamp

开始读取的时间（以 Unix 时间秒为单位） 在 StartingPosition 被指定为 AT_TIMESTAMP 的情况下定义 StartingPositionTimestamp。

类型：双精度

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给AWS::Lambda::EventSourceMapping资源的StartingPositionTimestamp属性。

Stream

数据流的 Amazon 资源名称（ARN）或流使用者。

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::EventSourceMapping`资源的`EventSourceArn`属性。

`TumblingWindowInSeconds`

处理窗口的持续时间（以秒为单位）。有效范围为 1 到 900（15 分钟）。

有关更多信息，请参阅《Amazon Lambda 开发人员指南》的滚动窗口。

类型：整数

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::EventSourceMapping`资源的`TumblingWindowInSeconds`属性。

示例

MetricsConfig 事件

以下是一个资源示例，该资源使用该`MetricsConfig`属性捕获其事件源映射的每个处理阶段。

```
Resources:
  FilteredEventsFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: s3://sam-demo-bucket/metricsConfig.zip
      Handler: index.handler
      Runtime: nodejs16.x
      Events:
        KinesisStream:
          Type: Kinesis
          Properties:
            Stream: !GetAtt KinesisStream.Arn
            StartingPosition: LATEST
            MetricsConfig:
              Metrics:
              - EventCount
```

Kinesis 事件源

下面是一个 Kinesis 事件源。

YAML

```
Events:
  KinesisEvent:
    Type: Kinesis
    Properties:
      Stream: arn:aws:kinesis:us-east-1:123456789012:stream/my-stream
      StartingPosition: TRIM_HORIZON
      BatchSize: 10
      Enabled: false
      FilterCriteria:
        Filters:
          - Pattern: '{"key": ["val1", "val2"]}'
```

MQ

描述 `MQ` 事件源类型的对象。有关更多信息，请参阅《Amazon Lambda 开发人员指南》中结合 Amazon MQ 使用 Lambda。

Amazon Serverless Application Model (Amazon SAM) 生成一个 AWS::Lambda::EventSourceMapping设置此事件类型时的资源。

> **Note**
>
> 要在虚拟私有云（VPC）中建立 Amazon MQ 队列并连接到公共网络中的 Lambda 函数，函数的执行角色必须包含以下权限：
>
> - `ec2:CreateNetworkInterface`
> - `ec2:DeleteNetworkInterface`
> - `ec2:DescribeNetworkInterfaces`
> - `ec2:DescribeSecurityGroups`
> - `ec2:DescribeSubnets`
> - `ec2:DescribeVpcs`
>
> 有关更多信息，请参阅《Amazon Lambda 开发人员指南》中的执行角色权限。

语法

要在 Amazon SAM 模板中声明此实体，请使用以下语法。

YAML

```
BatchSize: Integer
Broker: String
DynamicPolicyName: Boolean
Enabled: Boolean
FilterCriteria: FilterCriteria
KmsKeyArn: String
MaximumBatchingWindowInSeconds: Integer
Queues: List
SecretsManagerKmsKeyId: String
SourceAccessConfigurations: List
```

属性

`BatchSize`

要在单个批次中检索的最大项目数。

类型：整数

必需：否

默认值：100

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::EventSourceMapping`资源的`BatchSize`属性。

最小值：1

最大值：10000

`Broker`

Amazon MQ 代理的 Amazon 资源名称（ARN）。

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::EventSourceMapping`资源的`EventSourceArn`属性。

`DynamicPolicyName`

默认情况下， Amazon Identity and Access Management (IAM) 策略名称是`SamAutoGeneratedAMQPolicy`为了向后兼容。指定 `true` 对 IAM policy 使用自动生成的名称。此名称包含 Amazon MQ 事件源逻辑 ID。

> **Note**
>
> 使用多个 Amazon MQ 事件源时，请指定 `true`，以避免重复的 IAM policy 名称。

类型：布尔值

必需：否

默认值：`false`

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`Enabled`

如果为 `true`，则事件源映射处于活动状态。要暂停轮询和调用，设置为 `false`。

类型：布尔值

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::EventSourceMapping`资源的`Enabled`属性。

`FilterCriteria`

定义用于确定 Lambda 是否应处理事件的条件的对象。有关更多信息，请参阅《Amazon Lambda 开发人员指南》中的 Amazon Lambda 事件筛选。

类型：FilterCriteria

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::EventSourceMapping`资源的`FilterCriteria`属性。

`KmsKeyArn`

用于加密与此事件相关信息的密钥的 Amazon 资源名称（ARN）。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::EventSourceMapping`资源的`KmsKeyArn`属性。

`MaximumBatchingWindowInSeconds`

在调用函数之前收集记录的最长时间（以秒为单位）。

类型：整数

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::EventSourceMapping`资源的`MaximumBatchingWindowInSeconds`属性。

`Queues`

要使用的 Amazon MQ 代理目的地队列的名称。

类型：列表

必需：是

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::EventSourceMapping`资源的`Queues`属性。

`SecretsManagerKmsKeyId`

来自的客户托管密钥的 Amazon Key Management Service (Amazon KMS) 密钥 ID Amazon Secrets Manager。将来自 Secrets Manager 的客户托管密钥与不包含 `kms:Decrypt` 权限的 Lambda 执行角色一起使用时，此属性是必需的。

此属性的值为 UUID。例如：`1abc23d4-567f-8ab9-cde0-1fab234c5d67`。

类型：字符串

必需：条件

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`SourceAccessConfigurations`

身份验证协议或虚拟主机的数组。使用SourceAccessConfigurations数据类型进行指定。

对于 MQ 事件源类型，唯一有效的配置类型是 `BASIC_AUTH` 和 `VIRTUAL_HOST`。

- **`BASIC_AUTH`** – 存储您的代理凭证的 Secrets Manager 密钥。要使用此类型，凭证必须采用以下格式：`{"username": "your-username", "password": "your-password"}`。只允许使用一个类型为 `BASIC_AUTH` 的对象。
- **`VIRTUAL_HOST`** - 您的 RabbitMQ 代理中虚拟主机的名称。Lambda 将使用此 RabbitMQ 的主机作为事件源。只允许使用一个类型为 `VIRTUAL_HOST` 的对象。

类型：列表

必需：是

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::EventSourceMapping`资源的`SourceAccessConfigurations`属性。

示例

Amazon MQ 事件源

以下是 Amazon MQ 代理的 MQ 事件源类型示例。

YAML

```
Events:
  MQEvent:
    Type: MQ
    Properties:
      Broker: arn:aws:mq:us-
east-2:123456789012:broker:MyBroker:b-1234a5b6-78cd-901e-2fgh-3i45j6k178l9
      Queues: List of queues
      SourceAccessConfigurations:
        - Type: BASIC_AUTH
          URI: arn:aws:secretsmanager:us-east-1:01234567890:secret:MyBrokerSecretName
      BatchSize: 200
      Enabled: true
```

MSK

描述 MSK 事件源类型的对象。有关更多信息，请参阅Amazon Lambda 开发者指南中的Amazon Lambda 与 Amazon MSK 搭配使用。

Amazon Serverless Application Model (Amazon SAM) 在设置此事件类型时生成AWS::Lambda::EventSourceMapping资源。

要使用架构注册表，您需要为函数定义特定的 IAM 角色权限。有关所需配置的示例，请参阅使用 IAM 角色完成设置。

语法

要在 Amazon SAM 模板中声明此实体，请使用以下语法。

YAML

```
ConsumerGroupId: String
DestinationConfig: DestinationConfig
FilterCriteria: FilterCriteria
KmsKeyArn: String
MaximumBatchingWindowInSeconds: Integer
ProvisionedPollerConfig: ProvisionedPollerConfig
SchemaRegistryConfig: SchemaRegistryConfig
SourceAccessConfigurations: SourceAccessConfigurations
StartingPosition: String
StartingPositionTimestamp: Double
Stream: String
Topics: List
```

属性

`ConsumerGroupId`

用于配置如何从 Kafka 主题中读取事件的字符串。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::EventSourceMapping`资源的`AmazonManagedKafkaConfiguration`属性。

`DestinationConfig`

一个配置对象，用于在 Lambda 处理事件后指定事件目的地。

使用此属性指定来自 Amazon MSK 事件源的失败调用的目的地。

类型：DestinationConfig

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::EventSourceMapping`资源的 `DestinationConfig`属性。

`FilterCriteria`

定义用于确定 Lambda 是否应处理事件的条件的对象。有关更多信息，请参阅《Amazon Lambda 开发人员指南》中的 Amazon Lambda 事件筛选。

类型：FilterCriteria

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::EventSourceMapping`资源的`FilterCriteria`属性。

`KmsKeyArn`

用于加密与此事件相关信息的密钥的 Amazon 资源名称（ARN）。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::EventSourceMapping`资源的`KmsKeyArn`属性。

`MaximumBatchingWindowInSeconds`

在调用函数之前收集记录的最长时间（以秒为单位）。

类型：整数

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::EventSourceMapping`资源的`MaximumBatchingWindowInSeconds`属性。

`ProvisionedPollerConfig`

用于增加用于计算事件源映射的轮询器数量的配置。此配置允许最少 1 个轮询器和最多 20 个轮询器。有关示例，请参阅ProvisionedPollerConfig 示例。

类型：ProvisionedPollerConfig

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::EventSourceMapping`资源的`ProvisionedPollerConfig`属性。

`SchemaRegistryConfig`

用于在 Kafka 事件源中使用架构注册表的配置。

> **Note**
>
> 需要配置`ProvisionedPollerConfig`此功能。

类型: SchemaRegistryConfig

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::EventSourceMapping`资源的`AmazonManagedKafkaEventSourceConfig`属性。

`SourceAccessConfigurations`

用于保护与定义事件源的身份验证协议数组 VPC 组件或虚拟化主机。

有效值：`CLIENT_CERTIFICATE_TLS_AUTH`

类型：SourceAccessConfiguration 列表

必需：否

Amazon CloudFormation 兼容性：此属性是`AWS::Lambda::EventSourceMapping`资源AmazonManagedKafkaEventSourceConfig属性的一部分。

`StartingPosition`

在流中开始读取数据的位置。

- `AT_TIMESTAMP` – 指定开始读取记录的时间。

- LATEST - 仅读取新记录。
- TRIM_HORIZON - 处理所有可用的记录。

有效值：AT_TIMESTAMP | LATEST | TRIM_HORIZON

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给AWS::Lambda::EventSourceMapping资源的StartingPosition属性。

StartingPositionTimestamp

开始读取的时间（以 Unix 时间秒为单位）在 StartingPosition 被指定为 AT_TIMESTAMP 的情况下定义 StartingPositionTimestamp。

类型：双精度

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给AWS::Lambda::EventSourceMapping资源的StartingPositionTimestamp属性。

Stream

数据流的 Amazon 资源名称（ARN）或流使用者。

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性直接传递给AWS::Lambda::EventSourceMapping资源的EventSourceArn属性。

Topics

Kafka 主题的名称。

类型：列表

必需：是

Amazon CloudFormation 兼容性：此属性直接传递给AWS::Lambda::EventSourceMapping资源的Topics属性。

示例

使用 IAM 角色完成设置

以下示例显示了完整的设置，包括使用架构注册表所需的 IAM 角色配置：

```
Parameters:
  PreCreatedSubnetOne:
    Type: String
  PreCreatedSubnetTwo:
    Type: String
  MskClusterName4:
    Type: String

Resources:
  MyLambdaExecutionRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
        - Action: [sts:AssumeRole]
          Effect: Allow
          Principal:
            Service: [lambda.amazonaws.com]
      Policies:
      - PolicyName: KafkaClusterPermissions
        PolicyDocument:
          Statement:
          - Action: [kafka:DescribeClusterV2, kafka:GetBootstrapBrokers]
            Effect: Allow
            Resource: 'arn:aws:kafka:us-east-1:123456789012:cluster/*'
      - PolicyName: KafkaAuthPolicy
        PolicyDocument:
          Statement:
          - Action: [secretsmanager:GetSecretValue, kms:Decrypt]
            Effect: "Allow"
            Resource: ['arn:aws:secretsmanager:us-
west-2:123456789012:secret:kafkaSecret-******',
                        'arn:aws:kms:us-west-2:123456789012:key/keyId']
      - PolicyName: ENIPolicy
        PolicyDocument:
          Statement:
          - Action: [ec2:CreateNetworkInterface,
```

```
              ec2:DescribeNetworkInterfaces, ec2:DescribeVpcs,
 ec2:DeleteNetworkInterface,
              ec2:DescribeSubnets, ec2:DescribeSecurityGroups]
            Effect: Allow
            Resource: '*'
      - PolicyName: SchemaRegistryPolicy
        PolicyDocument:
          Statement:
          - Action: [glue:GetRegistry]
            Effect: Allow
            Resource: 'arn:aws:glue:{region}:{account-id}:registry/{registry-name}'
      - PolicyName: SchemaVersionsPolicy
        PolicyDocument:
          Statement:
          - Action: [glue:GetSchemaVersions]
            Effect: Allow
            Resource: '*'
      ManagedPolicyArns:
      - !Sub arn:${AWS::Partition}:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole
      Tags:
      - {Value: SAM, Key: lambda:createdBy}

  MyMskCluster:
    Type: AWS::MSK::Cluster
    Properties:
      BrokerNodeGroupInfo:
        ClientSubnets:
        - Ref: PreCreatedSubnetOne
        - Ref: PreCreatedSubnetTwo
        InstanceType: kafka.t3.small
        StorageInfo:
          EBSStorageInfo:
            VolumeSize: 1
      ClusterName:
        Ref: MskClusterName4
      KafkaVersion: 3.8.x
      NumberOfBrokerNodes: 2

  MyMskStreamProcessor:
    Type: AWS::Serverless::Function
    Properties:
      Runtime: nodejs18.x
      Handler: index.handler
```

```
      CodeUri: ${codeuri}
      Role:
        Fn::GetAtt: [MyLambdaExecutionRole, Arn]
      Events:
        MyMskEvent:
          Type: MSK
          Properties:
            StartingPosition: LATEST
            Stream:
              Ref: MyMskCluster
            SourceAccessConfigurations:
            - Type: SASL_SCRAM_512_AUTH
              URI: !Sub arn:${AWS::Partition}:secretsmanager:us-
west-2:123456789012:secret:my-path/my-secret-name-1a2b3c
            Topics:
            - SchemaRegistryTestTopic
            ProvisionedPollerConfig:
              MinimumPollers: 1
            SchemaRegistryConfig:
              AccessConfigs:
              - Type: BASIC_AUTH
                URI: !Sub arn:${AWS::Partition}:secretsmanager:us-
west-2:123456789012:secret:my-path/my-secret-name-1a2b3c
              SchemaValidationConfigs:
              - Attribute: KEY
              EventRecordFormat: JSON
              SchemaRegistryURI: !Sub arn:${AWS::Partition}:glue:us-
west-2:123456789012:registry/myregistry
```

ProvisionedPollerConfig 示例

```
ProvisionedPollerConfig:
  MinimumPollers: 1
  MaximumPollers: 20
```

现有集群的 Amazon MSK 示例

以下示例显示了 Amazon Web Services 账户中已存在的 Amazon MSK 集群的 `MSK` 事件源类型。

YAML

```
Events:
  MSKEvent:
```

```
    Type: MSK
    Properties:
      StartingPosition: LATEST
      Stream: arn:aws:kafka:us-east-1:012345678012:cluster/exampleClusterName/
abcdefab-1234-abcd-5678-cdef0123ab01-2
      Topics:
        - MyTopic
```

在同一模板中声明的集群的 Amazon MSK 示例

以下是在同一模板文件中声明的 Amazon MSK 集群的 MSK 事件源类型的示例。

YAML

```
Events:
  MSKEvent:
    Type: MSK
    Properties:
      StartingPosition: LATEST
      Stream:
        Ref: MyMskCluster   # This must be the name of an MSK cluster declared in the
 same template file
      Topics:
        - MyTopic
```

带有架构注册表的 MSK 事件源

以下是使用架构注册表配置MSK的事件源类型的示例。

```
Events:
  MSKEvent:
    Type: MSK
    Properties:
      StartingPosition: LATEST
      Stream:
        Ref: MyMskCluster
      Topics:
        - SchemaRegistryTestTopic
      ProvisionedPollerConfig:
        MinimumPollers: 1
      SchemaRegistryConfig:
        SchemaRegistryURI: !Sub arn:${AWS::Partition}:glue:us-
west-2:123456789012:registry/myregistry
```

```
        EventRecordFormat: JSON
        SchemaValidationConfigs:
          - Attribute: KEY
          - Attribute: VALUE
```

带有 Confluent 架构注册表的 MSK 事件源

以下是使用 Confluent 架构注册表配置MSK的事件源类型的示例。

```
Events:
  MSKEvent:
    Type: MSK
    Properties:
      StartingPosition: LATEST
      Stream:
        Ref: MyMskCluster
      Topics:
        - SchemaRegistryTestTopic
      ProvisionedPollerConfig:
        MinimumPollers: 1
      SchemaRegistryConfig:
        SchemaRegistryURI: https://my-schema-registry.confluent.cloud
        AccessConfigs:
          - Type: BASIC_AUTH
            URI: !Sub arn:${AWS::Partition}:secretsmanager:us-
west-2:123456789012:secret:my-secret
        EventRecordFormat: JSON
        SchemaValidationConfigs:
          - Attribute: KEY
          - Attribute: VALUE
```

S3

描述 S3 事件源类型的对象。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
  Bucket: String
  Events: String | List
```

```
  Filter: NotificationFilter
```

属性

`Bucket`

S3 桶名称。此存储桶必须存在于同一模板中。

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性类似于`AWS::S3::Bucket`资源的`BucketName`属性。在 SAM 中此为必填字段。此字段仅接受对此模板中创建的 S3 存储桶的引用

`Events`

调用 Lambda 函数的 Amazon S3 桶事件。有关有效值的列表，请参阅 Amazon S3 支持的事件类型。

类型：字符串 | 列表

必需：是

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::S3::BucketLambdaConfiguration`数据类型的`Event`属性。

`Filter`

确定哪些 Amazon S3 对象调用 Lambda 函数的筛选规则。有关 Amazon S3 键名筛选的信息，请参阅《Amazon Simple Storage Service 用户指南》中的配置 Amazon S3 事件通知。

类型：NotificationFilter

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::S3::BucketLambdaConfiguration`数据类型的`Filter`属性。

示例

S3 事件

S3 事件示例。

YAML

```
Events:
  S3Event:
    Type: S3
    Properties:
      Bucket:
        Ref: ImagesBucket     # This must be the name of an S3 bucket declared in the
 same template file
      Events: s3:ObjectCreated:*
      Filter:
        S3Key:
          Rules:
          - Name: prefix      # or "suffix"
            Value: value      # The value to search for in the S3 object key names
```

Schedule

描述`Schedule`事件源类型的对象，它将您的无服务器函数设置为按计划触发的 Amazon EventBridge 规则的目标。有关更多信息，请参阅什么是亚马逊 EventBridge？ 在《亚马逊 EventBridge 用户指南》中。

Amazon Serverless Application Model (Amazon SAM) 生成一个 AWS::Events::Rule设置此事件类型时的资源。

> **Note**
>
> EventBridge 现在提供了新的日程安排功能，Amazon EventBridge Scheduler。 亚马逊 EventBridge Scheduler 是一个无服务器调度程序，允许您通过一个中央托管服务创建、运行和管理任务。 EventBridge Scheduler 具有高度可定制性，与 EventBridge 计划规则相比，可扩展性更高，具有更广泛的目标 API 操作和 Amazon Web Services 服务。
> 我们建议您使用 EventBridge Scheduler 按计划调用目标。要在 Amazon SAM 模板中定义此事件源类型，请参阅ScheduleV2。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
DeadLetterConfig: DeadLetterConfig
Description: String
Enabled: Boolean
Input: String
Name: String
RetryPolicy: RetryPolicy
Schedule: String
State: String
```

属性

`DeadLetterConfig`

配置亚马逊简单队列服务 (Amazon SQS) Simple Queue Service 队列，目标调用失败后 EventBridge 在该队列中发送事件。例如，当向不存在的 Lambda 函数发送事件时，或者没有足够的权限调用 Lambda 函数 EventBridge 时，调用可能会失败。有关更多信息，请参阅 A ma EventBridge zon 用户指南中的事件重试策略和使用死信队列。

> **Note**
>
> AWS::Serverless::Function 资源类型具有类似的数据类型 `DeadLetterQueue`，用于处理成功调用目标 Lambda 函数后发生的故障。这些类型的故障示例包括 Lambda 节流或 Lambda 目标函数返回的错误。有关函数 `DeadLetterQueue` 属性的更多信息，请参阅《Amazon Lambda 开发人员指南》中的死信队列。

类型：DeadLetterConfig

必需：否

Amazon CloudFormation 兼容性：此属性类似于`AWS::Events::RuleTarget`数据类型的`DeadLetterConfig`属性。此属性的 Amazon SAM 版本包括其他子属性， Amazon SAM 以备您想要创建死信队列时使用。

`Description`

规则的描述。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Events::Rule`资源的`Description`属性。

`Enabled`

指示是否启用规则。

要禁用该规则，请将此属性设置为 `false`。

> **Note**
>
> 指定 `Enabled` 或 `State` 属性，但不能同时指定两者。

类型：布尔值

必需：否

Amazon CloudFormation 兼容性：此属性类似于`AWS::Events::Rule`资源的`State`属性。如果此属性设置为，`true`则 Amazon SAM 通过`ENABLED`，否则通过`DISABLED`。

`Input`

传递到目标的有效 JSON 文本。如果使用此属性，则不会将事件文本本身的任何内容传递到目标。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Events::Rule Target`资源的`Input`属性。

`Name`

规则的名称。如果您不指定名称，则 Amazon CloudFormation 会生成一个唯一的物理 ID 并将该 ID 用作规则名称。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Events::Rule`资源的`Name`属性。

`RetryPolicy`

包含有关重试策略设置的信息的 `RetryPolicy` 对象。有关更多信息，请参阅 A ma EventBridge zon 用户指南中的事件重试策略和使用死信队列。

类型：RetryPolicy

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Events::RuleTarget`数据类型的`RetryPolicy`属性。

`Schedule`

决定运行规则的时间和频率的计划表达式。有关更多信息，请参阅规则的计划表达式。

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Events::Rule`资源的`ScheduleExpression`属性。

`State`

规则的状态。

接受的值：`DISABLED | ENABLED`

> **Note**
>
> 指定 `Enabled` 或 `State` 属性，但不能同时指定两者。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Events::Rule`资源的`State`属性。

示例

CloudWatch 安排活动

CloudWatch 安排活动示例

YAML

```
CWSchedule:
  Type: Schedule
  Properties:
    Schedule: 'rate(1 minute)'
    Name: TestSchedule
    Description: test schedule
    Enabled: false
```

DeadLetterConfig

用于指定亚马逊简单队列服务 (Amazon SQS) Simple Queue Service 队列的对象，目标调用失败后 EventBridge 在该队列中发送事件。例如，在向不存在的 Lambda 函数发送事件或调用 Lambda 函数的权限不足时，调用可能会失败。有关更多信息，请参阅 A ma EventBridge zon 用户指南中的事件重试策略和使用死信队列。

> **Note**
>
> AWS::Serverless::Function 资源类型具有类似的数据类型 `DeadLetterQueue`，用于处理成功调用目标 Lambda 函数后发生的故障。此类故障的示例包括 Lambda 节流或 Lambda 目标函数返回的错误。有关函数 `DeadLetterQueue` 属性的更多信息，请参阅《Amazon Lambda 开发人员指南》中的死信队列。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
  Arn: String
  QueueLogicalId: String
  Type: String
```

属性

`Arn`

指定作为死信队列目标的 Amazon SQS 队列的 Amazon 资源名称（ARN）。

> **Note**
> 指定 Type 属性或 Arn 属性，但不能同时指定两者。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给AWS::Events::RuleDeadLetterConfig数据类型的Arn属性。

QueueLogicalId

指定了 Amazon SAM 创建的死信队列的自定义名称。Type

> **Note**
> 如果未设置 Type 属性，则将忽略该属性。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

Type

队列的类型。设置此属性后， Amazon SAM 会自动创建死信队列并附加必要的基于资源的策略，以授予规则资源向队列发送事件的权限。

> **Note**
> 指定 Type 属性或 Arn 属性，但不能同时指定两者。

有效值：SQS

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

示例

DeadLetterConfig

DeadLetterConfig

YAML

```
DeadLetterConfig:
  Type: SQS
  QueueLogicalId: MyDLQ
```

ScheduleV2

描述`ScheduleV2`事件源类型的对象，它将您的无服务器函数设置为按计划触发的 Amazon S EventBridge cheduler 事件的目标。有关更多信息，请参阅什么是 Amazon EventBridge 日程安排？在《EventBridge 日程安排器用户指南》中。

Amazon Serverless Application Model (Amazon SAM) 生成一个 AWS::Scheduler::Schedule设置此事件类型时的资源。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
DeadLetterConfig: DeadLetterConfig
Description: String
EndDate: String
FlexibleTimeWindow: FlexibleTimeWindow
GroupName: String
Input: String
KmsKeyArn: String
Name: String
```

```
OmitName: Boolean
PermissionsBoundary: String
RetryPolicy: RetryPolicy
RoleArn: String
ScheduleExpression: String
ScheduleExpressionTimezone: String
StartDate: String
State: String
```

属性

DeadLetterConfig

配置亚马逊简单队列服务 (Amazon SQS) Simple Queue Service 队列，目标调用失败后 EventBridge 在该队列中发送事件。例如，当向不存在的 Lambda 函数发送事件时，或者没有足够的权限调用 Lambda 函数 EventBridge 时，调用可能会失败。有关更多信息，请参阅《日 EventBridge 程安排器用户指南》中的为调度程序配置死信队列。EventBridge

> **Note**
>
> AWS::Serverless::Function 资源类型具有类似的数据类型 `DeadLetterQueue`，用于处理成功调用目标 Lambda 函数后发生的故障。这些类型的故障示例包括 Lambda 节流或 Lambda 目标函数返回的错误。有关函数 `DeadLetterQueue` 属性的更多信息，请参阅《Amazon Lambda 开发人员指南》中的死信队列。

类型：DeadLetterConfig

必需：否

Amazon CloudFormation 兼容性：此属性类似于`AWS::Scheduler::ScheduleTarget`数据类型的`DeadLetterConfig`属性。此属性的 Amazon SAM 版本包括其他子属性， Amazon SAM 以备您想要创建死信队列时使用。

Description

计划的描述。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Scheduler::Schedule`资源的`Description`属性。

`EndDate`

以 UTC 为单位的日期，在此日期之前计划可以调用其目标。根据计划的重复表达式，调用可能会在您指定的 EndDate 或之前停止。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Scheduler::Schedule`资源的`EndDate`属性。

`FlexibleTimeWindow`

允许配置可以调用计划的窗口。

类型：FlexibleTimeWindow

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Scheduler::Schedule`资源的`FlexibleTimeWindow`属性。

`GroupName`

将与此计划关联的计划组名称。如果未定义，则使用默认组。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Scheduler::Schedule`资源的`GroupName`属性。

`Input`

传递到目标的有效 JSON 文本。如果使用此属性，则不会将事件文本本身的任何内容传递到目标。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Scheduler::Schedule Target`资源的`Input`属性。

`KmsKeyArn`

将用于加密客户数据的 KMS 密钥的 ARN。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Scheduler::Schedule`资源的`KmsKeyArn`属性。

`Name`

计划的名称。如果您未指定名称，则 Amazon SAM 会生成格式为的名称，*`Function-Logical-IDEvent-Source-Name`*并使用该 ID 作为计划名称。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Scheduler::Schedule`资源的`Name`属性。

`OmitName`

默认情况下， Amazon SAM 生成并使用格式为的计划名称*`<Function-logical-ID><event-source-name>`*。将此属性设置`true`为 Amazon CloudFormation 生成唯一的物理 ID，然后改用该物理 ID 作为计划名称。

类型：布尔值

必需：否

默认值：`false`

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`PermissionsBoundary`

用于为角色设置权限边界的策略的 ARN。

> **Note**
>
> 如果已定义，`PermissionsBoundary`则 Amazon SAM 将对计划程序计划的目标 IAM 角色应用相同的边界。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::IAM::Role`资源的`PermissionsBoundary`属性。

`RetryPolicy`

包含有关重试策略设置的信息的 RetryPolicy 对象。

类型：RetryPolicy

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Scheduler::ScheduleTarget`数据类型的`RetryPolicy`属性。

`RoleArn`

调用计划时， EventBridge 计划程序将用于目标的 IAM 角色的 ARN。

类型：RoleArn

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Scheduler::ScheduleTarget`数据类型的`RoleArn`属性。

`ScheduleExpression`

决定运行调度器计划事件的时间和频率的计划表达式。

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Scheduler::Schedule`资源的`ScheduleExpression`属性。

`ScheduleExpressionTimezone`

评估计划表达式的时区。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Scheduler::Schedule`资源的`ScheduleExpressionTimezone`属性。

`StartDate`

以 UTC 为单位的日期，在此日期之后计划可以调用目标。根据计划的重复表达式，调用可能会在您指定的 StartDate 或之后发生。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Scheduler::Schedule`资源的`StartDate`属性。

`State`

调度器计划的状态。

接受的值：`DISABLED | ENABLED`

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Scheduler::Schedule`资源的`State`属性。

示例

定义 ScheduleV2 资源的基本示例

```
Resources:
  Function:
```

```
    Properties:
      ...
      Events:
        ScheduleEvent:
          Type: ScheduleV2
          Properties:
            ScheduleExpression: "rate(1 minute)"
        ComplexScheduleEvent:
          Type: ScheduleV2
          Properties:
            ScheduleExpression: rate(1 minute)
            FlexibleTimeWindow:
              Mode: FLEXIBLE
              MaximumWindowInMinutes: 5
            StartDate: '2022-12-28T12:00:00.000Z'
            EndDate: '2023-01-28T12:00:00.000Z'
            ScheduleExpressionTimezone: UTC
            RetryPolicy:
              MaximumRetryAttempts: 5
              MaximumEventAgeInSeconds: 300
            DeadLetterConfig:
              Type: SQS
```

> **Note**
>
> 生成的 ScheduleV2 物理 ID 不包括堆栈名称。

SelfManagedKafka

描述 `SelfManagedKafka` 事件源类型的对象。有关更多信息，请参阅《开发人员指南》中的Amazon Lambda 与自行管理的 Apache Kafka 配合使用。Amazon Lambda

Amazon Serverless Application Model (Amazon SAM) 在设置此事件类型时生成AWS::Lambda::EventSourceMapping资源。

要使用架构注册表，您需要为函数定义特定的 IAM 角色权限。有关所需配置的示例，请参阅使用 IAM 角色完成设置。

语法

要在 Amazon SAM 模板中声明此实体，请使用以下语法。

YAML

```
BatchSize: Integer
ConsumerGroupId: String
DestinationConfig: DestinationConfig
Enabled: Boolean
FilterCriteria: FilterCriteria
KafkaBootstrapServers: List
KmsKeyArn: String
ProvisionedPollerConfig: ProvisionedPollerConfig
SchemaRegistryConfig: SchemaRegistryConfig
SourceAccessConfigurations: SourceAccessConfigurations
StartingPosition: String
StartingPositionTimestamp: Double
Topics: List
```

属性

BatchSize

Lambda 从流中提取并发送到函数的每个批处理中的最大记录数。

类型：整数

必需：否

默认值：100

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::EventSourceMapping`资源的`BatchSize`属性。

最小值：1

最大值：10000

ConsumerGroupId

用于配置如何从 Kafka 主题中读取事件的字符串。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::EventSourceMapping`资源的`SelfManagedKafkaConfiguration`属性。

`DestinationConfig`

一个配置对象，用于在 Lambda 处理事件后指定事件目的地。

使用此属性指定来自管理 Kafka 事件源的失败调用的目的地。

类型：DestinationConfig

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::EventSourceMapping`资源的 `DestinationConfig`属性。

`Enabled`

禁用事件源映射以暂停轮询和调用。

类型：布尔值

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::EventSourceMapping`资源的`Enabled`属性。

`FilterCriteria`

定义用于确定 Lambda 是否应处理事件的条件的对象。有关更多信息，请参阅《Amazon Lambda 开发人员指南》中的 Amazon Lambda 事件筛选。

类型：FilterCriteria

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::EventSourceMapping`资源的`FilterCriteria`属性。

`KafkaBootstrapServers`

Kafka 代理的引导服务器列表。包括端口，例如 `broker.example.com:`*`xxxx`*

类型：列表

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`KmsKeyArn`

用于加密与此事件相关信息的密钥的 Amazon 资源名称（ARN）。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::EventSourceMapping`资源的`KmsKeyArn`属性。

`ProvisionedPollerConfig`

用于增加用于计算事件源映射的轮询器数量的配置。此配置允许最少 1 个轮询器和最多 20 个轮询器。有关示例，请参阅 ProvisionedPollerConfig 示例

类型：ProvisionedPollerConfig

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::EventSourceMapping`资源的`ProvisionedPollerConfig`属性。

`SchemaRegistryConfig`

用于将架构注册表与自管理 Kafka 事件源配合使用的配置。

> **Note**
>
> 需要配置`ProvisionedPollerConfig`此功能。

类型: SchemaRegistryConfig

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::EventSourceMapping`资源的`SelfManagedKafkaEventSourceConfig`属性。

`SourceAccessConfigurations`

用于保护与定义事件源的身份验证协议数组 VPC 组件或虚拟化主机。

有效值：`BASIC_AUTH | CLIENT_CERTIFICATE_TLS_AUTH | SASL_SCRAM_256_AUTH | SASL_SCRAM_512_AUTH | SERVER_ROOT_CA_CERTIFICATE`

类型：SourceAccessConfiguration 列表

必需：是

Amazon CloudFormation 兼容性：此属性是AWS::Lambda::EventSourceMapping资源SelfManagedKafkaEventSourceConfig属性的一部分。

StartingPosition

在流中开始读取数据的位置。

- AT_TIMESTAMP – 指定开始读取记录的时间。
- LATEST - 仅读取新记录。
- TRIM_HORIZON - 处理所有可用的记录。

有效值：AT_TIMESTAMP | LATEST | TRIM_HORIZON

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给AWS::Lambda::EventSourceMapping资源的StartingPosition属性。

StartingPositionTimestamp

开始读取的时间（以 Unix 时间秒为单位）在 StartingPosition 被指定为 AT_TIMESTAMP 的情况下定义 StartingPositionTimestamp。

类型：双精度

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给AWS::Lambda::EventSourceMapping资源的StartingPositionTimestamp属性。

Topics

Kafka 主题的名称。

类型：列表

必需：是

Amazon CloudFormation 兼容性：此属性直接传递给AWS::Lambda::EventSourceMapping资源的Topics属性。

示例

使用 IAM 角色完成设置

以下示例显示了完整的设置，包括使用架构注册表所需的 IAM 角色配置：

```
Parameters:
  PreCreatedSubnetOne:
    Type: String
  PreCreatedSubnetTwo:
    Type: String

Resources:
  MyLambdaExecutionRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
        - Action: [sts:AssumeRole]
          Effect: Allow
          Principal:
            Service: [lambda.amazonaws.com]
      Policies:
      - PolicyName: KafkaAuthPolicy
        PolicyDocument:
          Statement:
          - Action: [secretsmanager:GetSecretValue, kms:Decrypt]
            Effect: "Allow"
            Resource: ['arn:aws:secretsmanager:us-
west-2:123456789012:secret:kafkaSecret-******',
                        'arn:aws:kms:us-west-2:123456789012:key/keyId']
      - PolicyName: ENIPolicy
        PolicyDocument:
          Statement:
          - Action: [ec2:CreateNetworkInterface,
              ec2:DescribeNetworkInterfaces, ec2:DescribeVpcs,
 ec2:DeleteNetworkInterface,
              ec2:DescribeSubnets, ec2:DescribeSecurityGroups]
            Effect: Allow
            Resource: '*'
      - PolicyName: SchemaRegistryPolicy
        PolicyDocument:
          Statement:
```

```
          - Action: [glue:GetRegistry]
            Effect: Allow
            Resource: 'arn:aws:glue:{region}:{account-id}:registry/{registry-name}'
      - PolicyName: SchemaVersionsPolicy
        PolicyDocument:
          Statement:
          - Action: [glue:GetSchemaVersions]
            Effect: Allow
            Resource: '*'
      ManagedPolicyArns:
      - !Sub arn:${AWS::Partition}:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole
      Tags:
      - {Value: SAM, Key: lambda:createdBy}

  MyKafkaProcessor:
    Type: AWS::Serverless::Function
    Properties:
      Runtime: nodejs18.x
      Handler: index.handler
      CodeUri: ${codeuri}
      Role:
        Fn::GetAtt: [MyLambdaExecutionRole, Arn]
      Events:
        SelfManagedKafkaEvent:
          Type: SelfManagedKafka
          Properties:
            KafkaBootstrapServers:
              - my-kafka-broker-1:9092
              - my-kafka-broker-2:9092
            Topics:
              - SchemaRegistryTestTopic
            StartingPosition: LATEST
            SourceAccessConfigurations:
              - Type: VPC_SUBNET
                URI: subnet:subnet-12345678
              - Type: VPC_SECURITY_GROUP
                URI: security_group:sg-12345678
              - Type: BASIC_AUTH
                URI: !Sub arn:${AWS::Partition}:secretsmanager:us-
west-2:123456789012:secret:my-path/my-secret-name-1a2b3c
            ProvisionedPollerConfig:
              MinimumPollers: 1
            SchemaRegistryConfig:
```

```
              AccessConfigs:
              - Type: BASIC_AUTH
                URI: !Sub arn:${AWS::Partition}:secretsmanager:us-
west-2:123456789012:secret:my-path/my-secret-name-1a2b3c
              SchemaValidationConfigs:
              - Attribute: KEY
              EventRecordFormat: JSON
              SchemaRegistryURI: !Sub arn:${AWS::Partition}:glue:us-
west-2:123456789012:registry/myregistry
```

ProvisionedPollerConfig 示例

```
ProvisionedPollerConfig:
  MinimumPollers: 1
  MaximumPollers: 20
```

自行管理的 Kafka 事件源

以下是 `SelfManagedKafka` 事件源类型的示例。

YAML

```
Events:
  SelfManagedKafkaEvent:
    Type: SelfManagedKafka
    Properties:
      BatchSize: 1000
      Enabled: true
      KafkaBootstrapServers:
        - abc.xyz.com:xxxx
      SourceAccessConfigurations:
        -  Type: BASIC_AUTH
           URI: arn:aws:secretsmanager:us-west-2:123456789012:secret:my-path/my-secret-
name-1a2b3c
      Topics:
        - MyKafkaTopic
```

带有 Amazon Glue 架构注册表的自管理 Kafka 事件源

以下是使用 Amazon Glue 架构注册表配置`SelfManagedKafka`的事件源类型的示例。

```
Events:
```

```
  SelfManagedKafkaEvent:
    Type: SelfManagedKafka
    Properties:
      KafkaBootstrapServers:
        - abc.xyz.com:9092
      Topics:
        - SchemaRegistryTestTopic
      StartingPosition: LATEST
      ProvisionedPollerConfig:
        MinimumPollers: 1
      SchemaRegistryConfig:
        SchemaRegistryURI: !Sub arn:${AWS::Partition}:glue:us-
west-2:123456789012:registry/myregistry
        EventRecordFormat: JSON
        SchemaValidationConfigs:
          - Attribute: KEY
          - Attribute: VALUE
      SourceAccessConfigurations:
        - Type: VPC_SUBNET
          URI: subnet:subnet-12345678
        - Type: VPC_SECURITY_GROUP
          URI: security_group:sg-12345678
```

带有 Confluent 架构注册表的自管理 Kafka 事件源

以下是使用 Confluent 架构注册表配置`SelfManagedKafka`的事件源类型的示例。

```
Events:
  SelfManagedKafkaEvent:
    Type: SelfManagedKafka
    Properties:
      KafkaBootstrapServers:
        - abc.xyz.com:9092
      Topics:
        - SchemaRegistryTestTopic
      StartingPosition: LATEST
      ProvisionedPollerConfig:
        MinimumPollers: 1
      SchemaRegistryConfig:
        SchemaRegistryURI: https://my-schema-registry.confluent.cloud
        AccessConfigs:
          - Type: BASIC_AUTH
            URI: !Sub arn:${AWS::Partition}:secretsmanager:us-
west-2:123456789012:secret:my-secret
```

```
        EventRecordFormat: JSON
        SchemaValidationConfigs:
          - Attribute: KEY
          - Attribute: VALUE
      SourceAccessConfigurations:
        - Type: VPC_SUBNET
          URI: subnet:subnet-12345678
        - Type: VPC_SECURITY_GROUP
          URI: security_group:sg-12345678
        - Type: BASIC_AUTH
          URI: !Sub arn:${AWS::Partition}:secretsmanager:us-
west-2:123456789012:secret:kafka-secret
```

SNS

描述 SNS 事件源类型的对象。

SAM 生成 AWS::SNS::Subscription设置此事件类型时的资源

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
  FilterPolicy: SnsFilterPolicy
  FilterPolicyScope: String
  RedrivePolicy: Json
  Region: String
  SqsSubscription: Boolean | SqsSubscriptionObject
  Topic: String
```

属性

`FilterPolicy`

分配给订阅的筛选策略 JSON。有关更多信息，请参阅GetSubscriptionAttributes《亚马逊简单通知服务 API 参考》。

类型：SnsFilterPolicy

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::SNS::Subscription`资源的`FilterPolicy`属性。

`FilterPolicyScope`

此属性允许您使用以下字符串值类型之一来选择筛选范围：

- `MessageAttributes` – 筛选条件应用于消息属性。
- `MessageBody` – 筛选条件应用于消息正文。

类型：字符串

必需：否

默认值：`MessageAttributes`

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::SNS::Subscription`资源的`FilterPolicyScope`属性。

`RedrivePolicy`

指定后，将无法传输的消息发送到指定的 Amazon SQS 死信队列。由于客户端错误（例如，在无法访问订阅的端点时）或服务器错误（例如，在支持订阅的端点的服务变得不可用时）而无法传输的消息将保留在死信队列中，以进行进一步分析或重新处理。

有关重新驱动策略和死信队列的更多信息，请参阅《Amazon Simple Queue Service 开发人员指南》中的 Amazon SQS 死信队列。

类型：Json

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::SNS::Subscription`资源的`RedrivePolicy`属性。

`Region`

对于跨区域订阅，为主题所在的区域。

如果未指定区域，则 CloudFormation 使用呼叫者的区域作为默认区域。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::SNS::Subscription`资源的`Region`属性。

`SqsSubscription`

将此属性设置为 true，或指定 `SqsSubscriptionObject` 以在 SQS 队列中启用批处理 SNS 主题通知。将此属性设置为 `true` 可创建新的 SQS 队列，而指定 `SqsSubscriptionObject` 则使用现有的 SQS 队列。

类型：布尔值 | SqsSubscriptionObject

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`Topic`

要订阅的主题的 ARN。

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::SNS::Subscription`资源的`TopicArn`属性。

示例

SNS 事件源示例

SNS 事件源示例

YAML

```
Events:
  SNSEvent:
    Type: SNS
    Properties:
      Topic: arn:aws:sns:us-east-1:123456789012:my_topic
      SqsSubscription: true
      FilterPolicy:
        store:
          - example_corp
```

```
        price_usd:
          - numeric:
              - ">="
              - 100
```

SqsSubscriptionObject

为 SNS 事件指定现有 SQS 队列选项

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
BatchSize: String
Enabled: Boolean
QueueArn: String
QueuePolicyLogicalId: String
QueueUrl: String
```

属性

`BatchSize`

要在 SQS 队列单个批次中检索的最大项目数。

类型：字符串

必需：否

默认值：10

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`Enabled`

禁用 SQS 事件源映射以暂停轮询和调用。

类型：布尔值

必需：否

默认值：True

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`QueueArn`

指定现有 SQS 队列 arn。

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`QueuePolicyLogicalId`

为资源提供一个自定义 LogicalID 名称。AWS::SQS::QueuePolicy

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`QueueUrl`

指定与 `QueueArn` 属性关联的队列 URL。

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

示例

SNS 事件的现有 SQS

添加现有 SQS 队列以订阅 SNS 主题的示例。

YAML

```
QueuePolicyLogicalId: CustomQueuePolicyLogicalId
```

```
QueueArn:
  Fn::GetAtt: MyCustomQueue.Arn
QueueUrl:
  Ref: MyCustomQueue
BatchSize: 5
```

SQS

描述 SQS 事件源类型的对象。有关更多信息，请参阅《Amazon Lambda 开发者指南》中的Amazon Lambda 与 Amazon SQS 配合使用。

SAM 生成 AWS::Lambda::EventSourceMapping设置此事件类型时的资源

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
  BatchSize: Integer
  Enabled: Boolean
  FilterCriteria: FilterCriteria
  FunctionResponseTypes: List
  KmsKeyArn: String
  MaximumBatchingWindowInSeconds: Integer
  MetricsConfig: MetricsConfig
  Queue: String
  ScalingConfig: ScalingConfig
```

属性

`BatchSize`

要在单个批次中检索的最大项目数。

类型：整数

必需：否

默认值：10

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::EventSourceMapping`资源的`BatchSize`属性。

最小值：1

最大值：10000

`Enabled`

禁用事件源映射以暂停轮询和调用。

类型：布尔值

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::EventSourceMapping`资源的`Enabled`属性。

`FilterCriteria`

定义用于确定 Lambda 是否应处理事件的条件的对象。有关更多信息，请参阅《Amazon Lambda 开发人员指南》中的 Amazon Lambda 事件筛选。

类型：FilterCriteria

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::EventSourceMapping`资源的`FilterCriteria`属性。

`FunctionResponseTypes`

当前应用于事件源映射的响应类型的列表。有关详细信息，请参阅《Amazon Lambda 开发人员指南》中的报告批处理项目失败。

有效值：`ReportBatchItemFailures`

类型：列表

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::EventSourceMapping`资源的`FunctionResponseTypes`属性。

`KmsKeyArn`

用于加密与此事件相关信息的密钥的 Amazon 资源名称（ARN）。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::EventSourceMapping`资源的`KmsKeyArn`属性。

`MaximumBatchingWindowInSeconds`

在调用函数之前收集记录的最长时间（以秒为单位）。

类型：整数

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::EventSourceMapping`资源的`MaximumBatchingWindowInSeconds`属性。

`MetricsConfig`

一种可选配置，用于获取捕获每个处理阶段的事件源映射的增强指标。有关示例，请参阅MetricsConfig 事件。

类型：MetricsConfig

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::EventSourceMapping`资源的`MetricsConfig`属性。

`Queue`

队列的 ARN。

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::EventSourceMapping`资源的`EventSourceArn`属性。

`ScalingConfig`

扩展 SQS 轮询器的配置，以控制调用速率并设置最大并发调用次数。

类型：`ScalingConfig`

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::EventSourceMapping`资源的 `ScalingConfig`属性。

示例

MetricsConfig 事件

以下是一个资源示例，该资源使用该`MetricsConfig`属性捕获其事件源映射的每个处理阶段。

```
Resources:
  FilteredEventsFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: s3://sam-demo-bucket/metricsConfig.zip
      Handler: index.handler
      Runtime: nodejs16.x
      Events:
        KinesisStream:
          Type: Kinesis
          Properties:
            Stream: !GetAtt KinesisStream.Arn
            StartingPosition: LATEST
            MetricsConfig:
              Metrics:
              - EventCount
```

基本 SQS 事件

```
Events:
  SQSEvent:
    Type: SQS
    Properties:
      Queue: arn:aws:sqs:us-west-2:012345678901:my-queue
      BatchSize: 10
      Enabled: false
      FilterCriteria:
        Filters:
          - Pattern: '{"key": ["val1", "val2"]}'
```

为您的 SQS 队列配置部分批量报告

```
Events:
```

```
  SQSEvent:
    Type: SQS
    Properties:
      Enabled: true
      FunctionResponseTypes:
        - ReportBatchItemFailures
      Queue: !GetAtt MySqsQueue.Arn
      BatchSize: 10
```

带有配置了扩展的 SQS 事件的 Lambda 函数

```
MyFunction:
  Type: AWS::Serverless::Function
  Properties:
    ...
    Events:
      MySQSEvent:
        Type: SQS
        Properties:
          ...
          ScalingConfig:
            MaximumConcurrency: 10
```

FunctionCode

Lambda 函数的部署程序包。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
  Bucket: String
  Key: String
  Version: String
```

属性

`Bucket`

与您的函数位于同一 Amazon 区域的 Amazon S3 存储桶。

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::FunctionCode`数据类型的`S3Bucket`属性。

`Key`

部署程序包的 Amazon S3 密钥。

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::FunctionCode`数据类型的`S3Key`属性。

`Version`

对于版本控制的对象，指要使用的部署程序包对象的版本。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::FunctionCode`数据类型的`S3ObjectVersion`属性。

示例

FunctionCode

`CodeUri`：函数代码示例

YAML

```
CodeUri:
  Bucket: sam-s3-demo-bucket-name
  Key: mykey-name
  Version: 121212
```

FunctionUrlConfig

使用指定的配置参数创建 Amazon Lambda 函数 URL。Lambda 函数 URL 是一个 HTTPS 端点，可用于调用函数。

默认情况下，您创建的函数 URL 使用 Lambda 函数的 `$LATEST` 版本。如果为 Lambda 函数指定 `AutoPublishAlias`，则端点会连接到指定的函数别名。

有关更多信息，请参阅Amazon Lambda 开发人员指南 URLs中的 Lambda 函数。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
AuthType: String
Cors: Cors
InvokeMode: String
```

属性

`AuthType`

函数 URL 的身份验证类型。要使用 Amazon Identity and Access Management (IAM) 来授权请求，请将设置为`AWS_IAM`。对于开放式访问，设置为 `NONE`。

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::Url`资源的`AuthType`属性。

`Cors`

适用于函数 URL 的 cross-origin resource sharing (CORS)（跨源资源共享）设置。

类型：Cors

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::Url`资源的`Cors`属性。

`InvokeMode`

将会调用函数 URL 的模式。要让函数在调用完成后返回响应，请设置为 `BUFFERED`。要让函数流式传输响应，请设置为 `RESPONSE_STREAM`。默认值为 `BUFFERED`。

有效值：`BUFFERED` 或 `RESPONSE_STREAM`

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::Url`资源的`InvokeMode`属性。

示例

函数 URL

以下示例创建了带函数 URL 的 Lambda 函数。函数 URL 使用 IAM 授权。

YAML

```
HelloWorldFunction:
  Type: AWS::Serverless::Function
  Properties:
    CodeUri: hello_world/
    Handler: index.handler
    Runtime: nodejs20.x
    FunctionUrlConfig:
      AuthType: AWS_IAM
      InvokeMode: RESPONSE_STREAM

Outputs:
  MyFunctionUrlEndpoint:
      Description: "My Lambda Function URL Endpoint"
      Value:
        Fn::GetAtt: HelloWorldFunctionUrl.FunctionUrl
```

AWS::Serverless::GraphQLApi

使用 Amazon Serverless Application Model (Amazon SAM) `AWS::Serverless::GraphQLApi` 资源类型创建和配置 Amazon AppSync GraphQL 您的无服务器应用程序的 API。

要了解更多信息 Amazon AppSync，请参阅什么是 Amazon AppSync？ 在《Amazon AppSync 开发人员指南》中。

语法

YAML

```
LogicalId:
  Type: AWS::Serverless::GraphQLApi
  Properties:
    ApiKeys: ApiKeys
    Auth: Auth
    Cache: AWS::AppSync::ApiCache
    DataSources: DataSource
    DomainName: AWS::AppSync::DomainName
    Functions: Function
    Logging: LogConfig
    Name: String
    Resolvers: Resolver
    SchemaInline: String
    SchemaUri: String
    Tags:
    - Tag
    XrayEnabled: Boolean
```

属性

`ApiKeys`

创建可用于执行的唯一密钥 GraphQL 需要 API 密钥的操作。

类型：ApiKeys

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`Auth`

为您配置身份验证 GraphQL API。

类型：身份验证

必需：是

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`Cache`

`CreateApiCache` 操作的输入。

类型：AWS::AppSync::ApiCache

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给AWS::AppSync::ApiCache资源。

`DataSources`

为中的函数创建 Amazon AppSync 要连接的数据源。 Amazon SAM 支持 Amazon DynamoDB Amazon Lambda 和数据源。

类型：DataSource

必需：是

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`DomainName`

为您的自定义域名 GraphQL API。

类型：AWS::AppSync::DomainName

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给AWS::AppSync::DomainName资源。Amazon SAM 自动生成AWS::AppSync::DomainNameApiAssociation资源。

`Functions`

在中配置函数 GraphQL APIs 执行某些操作。

类型：函数

必需：是

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`Logging`

为您配置 Amazon CloudWatch 日志 GraphQL API。

如果不指定此属性， Amazon SAM 将生成`CloudWatchLogsRoleArn`并设置以下值：

- `ExcludeVerboseContent: true`
- `FieldLogLevel: ALL`

要选择退出日志记录，请指定以下内容：

```
Logging: false
```

类型：LogConfig

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::AppSync::GraphQLApi`资源的`LogConfig`属性。

`LogicalId`

你的唯一名字 GraphQL API。

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::AppSync::GraphQLApi`资源的`Name`属性。

`Name`

你的名字 GraphQL API。指定此属性以覆盖 `LogicalId` 值。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::AppSync::GraphQLApi`资源的`Name`属性。

`Resolvers`

为你的字段配置解析器 GraphQL API。 Amazon SAM 支持 JavaScript 管道解析器。

类型：解析程序

必需：是

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`SchemaInline`

a 的文本表示形式 GraphQL 架构在 SDL 格式的日期和时间。

类型：字符串

必填：条件性。您必须指定 `SchemaInline` 或 `SchemaUri`。

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::AppSync::GraphQLSchema`资源的`Definition`属性。

`SchemaUri`

架构的 Amazon Simple Storage Service (Amazon S3) 存储桶 URI 或本地文件夹的路径。

如果您指定本地文件夹的路径，则 Amazon CloudFormation 要求在部署之前先将文件上传到 Amazon S3。你可以使用 Amazon SAM CLI 以促进这一进程。有关更多信息，请参阅 如何在 Amazon SAM 部署时上传本地文件。

类型：字符串

必填：条件性。您必须指定 `SchemaInline` 或 `SchemaUri`。

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::AppSync::GraphQLSchema`资源的`DefinitionS3Location`属性。

`Tags`

这个的标签（键值对） GraphQL API。使用标签标识和分类资源。

类型：标签列表

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::AppSync::GraphQLApi`资源的`Tag`属性。

`XrayEnabled`

指明是否对此资源使用 Amazon X-Ray 跟踪。

类型：布尔值

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::AppSync::GraphQLApi`资源的`XrayEnabled`属性。

返回值

有关返回值的列表，请参阅 AWS::Serverless::GraphQLApi（在 Amazon CloudFormation 用户指南中）。

示例

GraphQL API 使用 DynamoDB 数据源

在这个例子中，我们创建了一个 GraphQL 使用 DynamoDB 表作为数据源的 API。

schema.graphql

```
schema {
  query: Query
  mutation: Mutation
}

type Query {
  getPost(id: String!): Post
}

type Mutation {
  addPost(author: String!, title: String!, content: String!): Post!
}

type Post {
  id: String!
  author: String
  title: String
  content: String
  ups: Int!
  downs: Int!
  version: Int!
}
```

template.yaml

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
...
Resources:
  DynamoDBPostsTable:
    Type: AWS::Serverless::SimpleTable

  MyGraphQLAPI:
    Type: AWS::Serverless::GraphQLApi
    Properties:
      SchemaUri: ./sam_graphql_api/schema.graphql
      Auth:
        Type: AWS_IAM
      DataSources:
        DynamoDb:
          PostsDataSource:
            TableName: !Ref DynamoDBPostsTable
            TableArn: !GetAtt DynamoDBPostsTable.Arn
      Functions:
        preprocessPostItem:
          Runtime:
            Name: APPSYNC_JS
            Version: 1.0.0
          DataSource: NONE
          CodeUri: ./sam_graphql_api/preprocessPostItem.js
        createPostItem:
          Runtime:
            Name: APPSYNC_JS
            Version: "1.0.0"
          DataSource: PostsDataSource
          CodeUri: ./sam_graphql_api/createPostItem.js
        getPostFromTable:
          Runtime:
            Name: APPSYNC_JS
            Version: "1.0.0"
          DataSource: PostsDataSource
          CodeUri: ./sam_graphql_api/getPostFromTable.js
      Resolvers:
        Mutation:
          addPost:
            Runtime:
              Name: APPSYNC_JS
```

```
              Version: "1.0.0"
            Pipeline:
            - preprocessPostItem
            - createPostItem
        Query:
          getPost:
            CodeUri: ./sam_graphql_api/getPost.js
            Runtime:
              Name: APPSYNC_JS
              Version: "1.0.0"
            Pipeline:
            - getPostFromTable
```

createPostItem.js

```
import { util } from "@aws-appsync/utils";

export function request(ctx) {
  const { key, values } = ctx.prev.result;
  return {
    operation: "PutItem",
    key: util.dynamodb.toMapValues(key),
    attributeValues: util.dynamodb.toMapValues(values),
  };
}

export function response(ctx) {
  return ctx.result;
}
```

getPostFromTable.js

```
import { util } from "@aws-appsync/utils";

export function request(ctx) {
  return dynamoDBGetItemRequest({ id: ctx.args.id });
}

export function response(ctx) {
  return ctx.result;
}

/**
```

```
 * A helper function to get a DynamoDB item
 */
function dynamoDBGetItemRequest(key) {
  return {
    operation: "GetItem",
    key: util.dynamodb.toMapValues(key),
  };
}
```

preprocessPostItem.js

```
import { util } from "@aws-appsync/utils";

export function request(ctx) {
  const id = util.autoId();
  const { ...values } = ctx.args;
  values.ups = 1;
  values.downs = 0;
  values.version = 1;
  return { payload: { key: { id }, values: values } };
}

export function response(ctx) {
  return ctx.result;
}
```

解析程序代码如下：

getPost.js

```
export function request(ctx) {
  return {};
}

export function response(ctx) {
  return ctx.prev.result;
}
```

GraphQL 以 Lambda 函数作为数据源的 API

在这个例子中，我们创建了一个 GraphQL 使用 Lambda 函数作为数据源的 API。

template.yaml

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
...
Resources:
  MyLambdaFunction:
    Type: AWS::Serverless::Function
    Properties:
      Handler: index.handler
      Runtime: nodejs20.x
      CodeUri: ./lambda

  MyGraphQLAPI:
    Type: AWS::Serverless::GraphQLApi
    Properties:
      Name: MyApi
      SchemaUri: ./gql/schema.gql
      Auth:
        Type: API_KEY
      ApiKeys:
        MyApiKey:
          Description: my api key
      DataSources:
        Lambda:
          MyLambdaDataSource:
            FunctionArn: !GetAtt MyLambdaFunction.Arn
      Functions:
        lambdaInvoker:
          Runtime:
            Name: APPSYNC_JS
            Version: 1.0.0
          DataSource: MyLambdaDataSource
          CodeUri: ./gql/invoker.js
      Resolvers:
        Mutation:
          addPost:
            Runtime:
              Name: APPSYNC_JS
              Version: 1.0.0
            Pipeline:
            - lambdaInvoker
        Query:
          getPost:
            Runtime:
```

```
              Name: APPSYNC_JS
              Version: 1.0.0
            Pipeline:
            - lambdaInvoker

Outputs:
  MyGraphQLAPI:
    Description: AppSync API
    Value: !GetAtt MyGraphQLAPI.GraphQLUrl
  MyGraphQLAPIMyApiKey:
    Description: API Key for authentication
    Value: !GetAtt MyGraphQLAPIMyApiKey.ApiKey
```

schema.graphql

```
schema {
  query: Query
  mutation: Mutation
}
type Query {
  getPost(id: ID!): Post
}
type Mutation {
  addPost(id: ID!, author: String!, title: String, content: String): Post!
}
type Post {
  id: ID!
  author: String!
  title: String
  content: String
  ups: Int
  downs: Int
}
```

函数如下：

lambda/index.js

```
exports.handler = async (event) => {
  console.log("Received event {}", JSON.stringify(event, 3));

  const posts = {
    1: {
```

```
      id: "1",
      title: "First book",
      author: "Author1",
      content: "Book 1 has this content",
      ups: "100",
      downs: "10",
    },
  };

  console.log("Got an Invoke Request.");
  let result;
  switch (event.field) {
    case "getPost":
      return posts[event.arguments.id];
    case "addPost":
      // return the arguments back
      return event.arguments;
    default:
      throw new Error("Unknown field, unable to resolve " + event.field);
  }
};
```

invoker.js

```
import { util } from "@aws-appsync/utils";

export function request(ctx) {
  const { source, args } = ctx;
  return {
    operation: "Invoke",
    payload: { field: ctx.info.fieldName, arguments: args, source },
  };
}

export function response(ctx) {
  return ctx.result;
}
```

ApiKeys

创建可用于执行的唯一密钥 GraphQL 需要 API 密钥的操作。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
LogicalId:
  ApiKeyId: String
  Description: String
  ExpiresOn: Double
```

属性

ApiKeyId

您的 API 密钥的唯一名称。指定以覆盖 `LogicalId` 值。

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::AppSync::ApiKey`资源的`ApiKeyId`属性。

Description

您的 API 键的描述。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::AppSync::ApiKey`资源的`Description`属性。

ExpiresOn

API 密钥到期的时间。日期表示为自纪元以来的秒数，向下舍入到最接近的小时。

类型：双精度

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::AppSync::ApiKey`资源的`Expires`属性。

`LogicalId`

您的 API 密钥的唯一名称。

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::AppSync::ApiKey`资源的`ApiKeyId`属性。

Auth

为您配置授权 GraphQL API。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
Additional:
- AuthProvider
LambdaAuthorizer: LambdaAuthorizerConfig
OpenIDConnect: OpenIDConnectConfig
Type: String
UserPool: UserPoolConfig
```

属性

`Additional`

您的其他授权类型列表 GraphQL API。

类型：清单 AuthProvider

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`LambdaAuthorizer`

为 Lambda 函数授权方指定可选的授权配置。如果 `Type` 被指定为 `AWS_LAMBDA`，则可以配置此可选属性。

类型:LambdaAuthorizerConfig

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::AppSync::GraphQLApi`资源的`LambdaAuthorizerConfig`属性。

`OpenIDConnect`

为你指定可选的授权配置 OpenID Connect 合规服务。如果 `Type` 被指定为 `OPENID_CONNECT`，则可以配置此可选属性。

类型：Open IDConnect Config

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::AppSync::GraphQLApi`资源的`OpenIDConnectConfig`属性。

`Type`

应用程序和您的应用程序之间的默认授权类型 Amazon AppSync GraphQL API。

有关允许值的列表和描述，请参阅《Amazon AppSync 开发人员指南》中的授权和身份验证。

当您指定 Lambda 授权方 (`AWS_LAMBDA`) 时， Amazon SAM 会创建一个 Amazon Identity and Access Management (IAM) 策略来配置您之间的权限 GraphQL API 和 Lambda 函数。

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::AppSync::GraphQLApi`资源的`AuthenticationType`属性。

`UserPool`

指定可选的授权配置，以使用 Amazon Cognito 用户群体。如果 `Type` 被指定为 `AMAZON_COGNITO_USER_POOLS`，则可以配置此可选属性。

类型:UserPoolConfig

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::AppSync::GraphQLApi`资源的`UserPoolConfig`属性。

示例

配置默认授权类型和其他授权类型

在此示例中，我们首先将 Lambda 授权方配置为我们的默认授权类型 GraphQL API。

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
...
Resources:
  MyGraphQLAPI:
    Type: AWS::Serverless::GraphQLApi
    Properties:
      Auth:
        Type: AWS_LAMBDA
        LambdaAuthorizer:
          AuthorizerUri: !GetAtt Authorizer1.Arn
          AuthorizerResultTtlInSeconds: 10
          IdentityValidationExpression: hello
```

接下来，我们为我们配置其他授权类型 GraphQL 通过在我们的 Amazon SAM 模板中添加以下内容来进行 API：

```
        Additional:
        - Type: AWS_IAM
        - Type: API_KEY
        - Type: OPENID_CONNECT
          OpenIDConnect:
            AuthTTL: 10
            ClientId: myId
            IatTTL: 10
            Issuer: prod
```

这将生成以下 Amazon SAM 模板：

```
AWSTemplateFormatVersion: '2010-09-09'
```

```
Transform: AWS::Serverless-2016-10-31
...
Resources:
  MyGraphQLAPI:
    Type: AWS::Serverless::GraphQLApi
    Properties:
      Auth:
        Type: AWS_LAMBDA
        LambdaAuthorizer:
          AuthorizerUri: !GetAtt Authorizer1.Arn
          AuthorizerResultTtlInSeconds: 10
          IdentityValidationExpression: hello
        Additional:
        - Type: AWS_IAM
        - Type: API_KEY
        - Type: OPENID_CONNECT
          OpenIDConnect:
            AuthTTL: 10
            ClientId: myId
            IatTTL: 10
            Issuer: prod
```

AuthProvider

为您的额外用户提供可选的授权配置 GraphQL API 授权类型。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
LambdaAuthorizer: LambdaAuthorizerConfig
OpenIDConnect: OpenIDConnectConfig
Type: String
UserPool: UserPoolConfig
```

属性

`LambdaAuthorizer`

为您的 Amazon Lambda 函数授权方指定可选的授权配置。如果 `Type` 被指定为 `AWS_LAMBDA`，则可以配置此可选属性。

类型: LambdaAuthorizerConfig

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::AppSync::GraphQLApi` `AdditionalAuthenticationProvider`对象的 `LambdaAuthorizerConfig`属性。

`OpenIDConnect`

为您指定可选的授权配置 OpenID Connect 合规服务。如果 `Type` 被指定为 `OPENID_CONNECT`，则可以配置此可选属性。

类型：Open IDConnect Config

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::AppSync::GraphQLApi` `AdditionalAuthenticationProvider`对象的 `OpenIDConnectConfig`属性。

`Type`

应用程序和您的应用程序之间的默认授权类型 Amazon AppSync GraphQL API。

有关允许值的列表和描述，请参阅《Amazon AppSync 开发人员指南》中的授权和身份验证。

当您指定 Lambda 授权方 (`AWS_LAMBDA`) 时， Amazon SAM 会创建一个 Amazon Identity and Access Management (IAM) 策略来配置您之间的权限 GraphQL API 和 Lambda 函数。

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::AppSync::GraphQLApi` `AdditionalAuthenticationProvider`对象的 `AuthenticationType`属性。

`UserPool`

指定可选的授权配置，以使用 Amazon Cognito 用户群体。如果 `Type` 被指定为 `AMAZON_COGNITO_USER_POOLS`，则可以配置此可选属性。

类型: UserPoolConfig

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::AppSync::GraphQLApi` `AdditionalAuthenticationProvider`对象的 `UserPoolConfig`属性。

DataSource

配置您的数据源 GraphQL API 解析器可以连接到。您可以使用 Amazon Serverless Application Model (Amazon SAM) 模板来配置与以下数据源的连接：

- Amazon DynamoDB
- Amazon Lambda

要了解有关数据源的更多信息，请参阅《Amazon AppSync 开发人员指南》中的附加数据源。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
DynamoDb: DynamoDb
Lambda: Lambda
```

属性

`DynamoDb`

将 DynamoDB 表配置为你的数据源 GraphQL API 解析器。

类型：DynamoDb

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`Lambda`

将 Lambda 函数配置为您的数据源 GraphQL API 解析器。

类型：Lambda

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

DynamoDb

将 Amazon DynamoDB 表配置为你的数据源 GraphQL API 解析器。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
LogicalId:
  DeltaSync: DeltaSyncConfig
  Description: String
  Name: String
  Permissions: List
  Region: String
  ServiceRoleArn: String
  TableArn: String
  TableName: String
  UseCallerCredentials: Boolean
  Versioned: Boolean
```

属性

`DeltaSync`

描述增量同步配置。

类型：DeltaSyncConfig

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::AppSync::DataSource` `DynamoDBConfig`对象的`DeltaSyncConfig`属性。

`Description`

数据源的描述。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::AppSync::DataSource`资源的`Description`属性。

`LogicalId`

数据源的唯一名称。

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::AppSync::DataSource`资源的`Name`属性。

`Name`

数据源的名称。指定此属性以覆盖 `LogicalId` 值。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::AppSync::DataSource`资源的`Name`属性。

`Permissions`

使用 Amazon SAM 连接器 配置对数据源的权限。您可以在列表中提供以下任何值：

- `Read` - 允许解析程序读取数据源。
- `Write` - 允许解析程序将数据写入到数据源。

Amazon SAM 使用在部署时转换的`AWS::Serverless::Connector`资源来配置您的权限。要了解有关生成的资源的信息，请参阅Amazon CloudFormation 指定时生成的资源 AWS::Serverless::Connector。

> **Note**
>
> 您可以指定 `Permissions` 或 `ServiceRoleArn`，但不能同时指定两者。如果两者都未指定，则 Amazon SAM 将生成默认值为 `Read` and `Write`。要撤消对数据源的访问权限，请从模板中移除 DynamoDB 对象。 Amazon SAM

类型：列表

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。它类似于 `AWS::Serverless::Connector` 资源的 `Permissions` 属性。

`Region`

你的 Amazon Web Services 区域 DynamoDB 表的。如果未指定，则 Amazon SAM 使用`AWS::Region`。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::AppSync::DataSource` `DynamoDBConfig`对象的`AwsRegion`属性。

`ServiceRoleArn`

数据 Amazon Identity and Access Management 源的 (IAM) 服务角色 ARN。在访问数据源时，系统将代入此角色。

您可以指定 `Permissions` 或 `ServiceRoleArn`，但不能同时指定两者。

类型：字符串

必填项：否。 如果未指定，则 Amazon SAM 应用的默认值`Permissions`。

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::AppSync::DataSource`资源的`ServiceRoleArn`属性。

`TableArn`

DynamoDB 表的 ARN。

类型：字符串

必填：条件性。如果未指定 `ServiceRoleArn`，则需要有 `TableArn`。

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`TableName`

表名称。

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::AppSync::DataSource DynamoDBConfig`对象的`TableName`属性。

`UseCallerCredentials`

设置为 `true` 以将 IAM 与此数据源一起使用。

类型：布尔值

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::AppSync::DataSource DynamoDBConfig`对象的`UseCallerCredentials`属性。

`Versioned`

设置为 `true` 以将冲突检测、冲突解决和同步与该数据源一起使用。

类型：布尔值

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::AppSync::DataSource DynamoDBConfig`对象的`Versioned`属性。

Lambda

将 Amazon Lambda 函数配置为您的数据源 GraphQL API 解析器。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
LogicalId:
```

```
  Description: String
  FunctionArn: String
  Name: String
  ServiceRoleArn: String
```

属性

Description

数据源的描述。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::AppSync::DataSource`资源的`Description`属性。

FunctionArn

Lambda 函数的 ARN。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::AppSync::DataSource` `LambdaConfig`对象的`LambdaFunctionArn`属性。

LogicalId

数据源的唯一名称。

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::AppSync::DataSource`资源的`Name`属性。

Name

数据源的名称。指定此属性以覆盖 `LogicalId` 值。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::AppSync::DataSource`资源的`Name`属性。

`ServiceRoleArn`

数据 Amazon Identity and Access Management 源的 (IAM) 服务角色 ARN。在访问数据源时，系统将代入此角色。

> **Note**
> 要撤销对数据源的访问权限，请从 Amazon SAM 模板中移除 Lambda 对象。

类型：字符串

必填项：否。 如果未指定， Amazon SAM 将使用配置`Write`权限Amazon SAM 连接器。

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::AppSync::DataSource`资源的`ServiceRoleArn`属性。

函数

在中配置函数 GraphQL APIs 执行某些操作。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```yaml
LogicalId:
  CodeUri: String
  DataSource: String
  Description: String
  Id: String
  InlineCode: String
  MaxBatchSize: Integer
  Name: String
  Runtime: Runtime
  Sync: SyncConfig
```

属性

`CodeUri`

函数代码的 Amazon Simple Storage Service (Amazon S3) URI 或本地文件夹路径。

如果您指定本地文件夹的路径，则 Amazon CloudFormation 要求在部署之前先将文件上传到 Amazon S3。你可以使用 Amazon SAM CLI 以促进这一进程。有关更多信息，请参阅 如何在 Amazon SAM 部署时上传本地文件。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::AppSync::FunctionConfiguration`资源的`CodeS3Location`属性。

`DataSource`

此函数将附加到的数据源的名称。

- 要引用 `AWS::Serverless::GraphQLApi` 资源中的数据源，请指定其逻辑 ID。
- 要引用 `AWS::Serverless::GraphQLApi` 资源之外的数据源，请使用 `Fn::GetAtt` 内置函数提供其 `Name` 属性。例如，`!GetAtt MyLambdaDataSource.Name`。
- 要引用其他堆栈中的数据源，请使用 `Fn::ImportValue`。

如果指定了变体，`[NONE | None | none]`则 Amazon SAM 将为该`AWS::AppSync::DataSourceType`对象生成一个`None`值。

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::AppSync::FunctionConfiguration`资源的`DataSourceName`属性。

`Description`

函数的描述。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::AppSync::FunctionConfiguration`资源的`Description`属性。

`Id`

位于 `AWS::Serverless::GraphQLApi` 资源外的函数的函数 ID。

- 要在同一 Amazon SAM 模板中引用函数，请使用`Fn::GetAtt`内部函数。例如 `Id: !GetAtt createPostItemFunc.FunctionId`。
- 要引用其他堆栈中的函数，请使用 `Fn::ImportValue`。

使用时`Id`，不允许使用所有其他属性。 Amazon SAM 将自动传递您引用的函数的函数 ID。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`InlineCode`

包含请求和响应函数的函数代码。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::AppSync::FunctionConfiguration`资源的`Code`属性。

`LogicalId`

函数的唯一名称。

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::AppSync::FunctionConfiguration`资源的`Name`属性。

`MaxBatchSize`

向 `BatchInvoke` 操作中单个 Amazon Lambda 函数发送的解析程序请求输入的最大数量。

类型：整数

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::AppSync::FunctionConfiguration`资源的`MaxBatchSize`属性。

`Name`

函数的名称。指定以覆盖 `LogicalId` 值。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::AppSync::FunctionConfiguration`资源的`Name`属性。

`Runtime`

描述 Amazon AppSync 管道解析器或 Amazon AppSync 函数使用的运行时。指定要使用的运行时的名称和版本。

类型：运行时

必需：是

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。它类似于 `AWS::AppSync::FunctionConfiguration` 资源的 `Runtime` 属性。

`Sync`

描述函数的同步配置。

指定在调用函数时要使用的冲突检测策略和解决策略。

类型：SyncConfig

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::AppSync::FunctionConfiguration`资源的`SyncConfig`属性。

运行时

管道解析程序或函数的运行时间。指定要使用的名称和版本。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
Name: String
Version: String
```

属性

`Name`

要使用的运行时系统的名称。当前，允许的唯一值为 `APPSYNC_JS`。

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::AppSync::FunctionConfiguration AppSyncRuntime`对象的`Name`属性。

`Version`

要使用的运行时系统的版本。当前允许的唯一版本为 `1.0.0`。

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::AppSync::FunctionConfiguration AppSyncRuntime`对象的`RuntimeVersion`属性。

解析程序

为你的字段配置解析器 GraphQL API。 Amazon Serverless Application Model (Amazon SAM) 支持JavaScript 管道解析器。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```yaml
OperationType:
  LogicalId:
    Caching: CachingConfig
    CodeUri: String
    FieldName: String
    InlineCode: String
    MaxBatchSize: Integer
    Pipeline: List
    Runtime: Runtime
    Sync: SyncConfig
```

属性

`Caching`

激活了缓存的解析程序的缓存配置。

类型：CachingConfig

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::AppSync::Resolver`资源的`CachingConfig`属性。

`CodeUri`

解析程序函数代码的 Amazon Simple Storage Service (Amazon S3) URI 或本地文件夹路径。

如果您指定本地文件夹的路径，则 Amazon CloudFormation 要求在部署之前先将文件上传到 Amazon S3。你可以使用 Amazon SAM CLI 以促进这一进程。有关更多信息，请参阅 如何在 Amazon SAM 部署时上传本地文件。

如果两者均未提供`CodeUri`或`InlineCode`，则 Amazon SAM 将生成`InlineCode`将请求重定向到第一个管道函数并接收来自最后一个管道函数的响应。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::AppSync::Resolver`资源的`CodeS3Location`属性。

`FieldName`

解析程序的名称。指定此属性以覆盖 `LogicalId` 值。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::AppSync::Resolver`资源的`FieldName`属性。

`InlineCode`

包含请求和响应函数的解析程序代码。

如果两者均未提供`CodeUri`或`InlineCode`，则 Amazon SAM 将生成`InlineCode`将请求重定向到第一个管道函数并接收来自最后一个管道函数的响应。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::AppSync::Resolver`资源的`Code`属性。

`LogicalId`

解析程序的唯一名称。在一个 GraphQL schema，您的解析器名称应与其使用的字段名称相匹配。对`LogicalId` 也使用这个字段名称。

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`MaxBatchSize`

向 `BatchInvoke` 操作中单个 Amazon Lambda 函数发送的解析程序请求输入的最大数量。

类型：整数

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::AppSync::Resolver`资源的`MaxBatchSize`属性。

`OperationType`

这些区域有：GraphQL 与您的解析器关联的操作类型。例如，`Query`、`Mutation` 或 `Subscription`。您可以按`LogicalId` 将多个解析程序嵌套在单个`OperationType`中。

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::AppSync::Resolver`资源的`TypeName`属性。

`Pipeline`

与管道解决程序关联的函数。在列表中按逻辑 ID 指定函数。

类型：列表

必需：是

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。它类似于 `AWS::AppSync::Resolver` 资源的 `PipelineConfig` 属性。

`Runtime`

管道解析程序或函数的运行时间。指定要使用的名称和版本。

类型：运行时

必需：是

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。它类似于 `AWS::AppSync::Resolver` 资源的 `Runtime` 属性。

`Sync`

描述解析程序的同步配置。

指定在调用解析程序时要使用的冲突检测策略和解决策略。

类型：SyncConfig

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::AppSync::Resolver`资源的`SyncConfig`属性。

示例

使用 Amazon SAM 生成的解析器函数代码并将字段另存为变量

这里是 GraphQL 我们示例的架构：

```
schema {
  query: Query
  mutation: Mutation
}

type Query {
  getPost(id: ID!): Post
}

type Mutation {
  addPost(author: String!, title: String!, content: String!): Post!
}

type Post {
  id: ID!
  author: String
  title: String
  content: String
}
```

以下是我们 Amazon SAM 模板的片段：

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
...
Resources:
  MyGraphQLApi:
    Type: AWS::Serverless::GraphQLApi
    Properties:
      ...
      Functions:
        preprocessPostItem:
          ...
        createPostItem:
          ...
      Resolvers:
        Mutation:
```

```
          addPost:
            Runtime:
              Name: APPSYNC_JS
              Version: 1.0.0
            Pipeline:
            - preprocessPostItem
            - createPostItem
```

在我们的 Amazon SAM 模板中，我们没有指定`CodeUri`或`InlineCode`。在部署时， Amazon SAM 会自动为我们的解析器生成以下内联代码：

```
export function request(ctx) {
  return {};
}

export function response(ctx) {
  return ctx.prev.result;
}
```

这个默认的解析程序代码会将请求重定向到第一个管道函数，并接收来自最后一个管道函数的响应。

在第一个管道函数中，可以使用提供的 `args` 字段来解析请求对象并创建变量。然后就可以在函数中使用这些变量。以下是 `preprocessPostItem` 函数的示例：

```
import { util } from "@aws-appsync/utils";

export function request(ctx) {
  const author = ctx.args.author;
  const title = ctx.args.title;
  const content = ctx.args.content;

  // Use variables to process data

}

export function response(ctx) {
  return ctx.result;
}
```

运行时

管道解析程序或函数的运行时间。指定要使用的名称和版本。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
Name: String
Version: String
```

属性

`Name`

要使用的运行时系统的名称。当前，允许的唯一值为 `APPSYNC_JS`。

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::AppSync::Resolver` `AppSyncRuntime`对象的`Name`属性。

`Version`

要使用的运行时系统的版本。当前允许的唯一版本为 `1.0.0`。

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::AppSync::Resolver` `AppSyncRuntime`对象的`RuntimeVersion`属性。

AWS::Serverless::HttpApi

创建 Amazon API Gateway HTTP API，这使您能够以比 REST 更低的延迟和更低的成本 RESTful APIs 进行创作 APIs。有关更多信息，请参阅《API Gateway 开发者指南》APIs中的 “使用 HTTP”。

我们建议您使用 Amazon CloudFormation 挂钩或 IAM 策略来验证 API Gateway 资源是否附加了授权者来控制对它们的访问。

有关使用 Amazon CloudFormation 挂钩的更多信息，请参阅 Amazon CloudFormation CLI 用户指南和apigw-enforce-authorizer GitHub 存储库中的注册挂钩。

有关使用 IAM 策略的更多信息，请参阅《API Gateway 开发人员指南》中的要求 API 路由具有授权。

> **Note**
>
> 部署到时 Amazon CloudFormation， Amazon SAM 会将您的 Amazon SAM 资源转换为 Amazon CloudFormation 资源。有关更多信息，请参阅 生成的 Amazon CloudFormation 资源用于 Amazon SAM。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
Type: AWS::Serverless::HttpApi
Properties:
  AccessLogSettings: AccessLogSettings
  Auth: HttpApiAuth
  CorsConfiguration: String | HttpApiCorsConfiguration
  DefaultRouteSettings: RouteSettings
  DefinitionBody: JSON
  DefinitionUri: String | HttpApiDefinition
  Description: String
  DisableExecuteApiEndpoint: Boolean
  Domain: HttpApiDomainConfiguration
  FailOnWarnings: Boolean
  Name: String
  PropagateTags: Boolean
  RouteSettings: RouteSettings
  StageName: String
  StageVariables: Json
  Tags: Map
```

属性

`AccessLogSettings`

某个阶段的访问日志记录的设置。

类型：AccessLogSettings

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::ApiGatewayV2::Stage`资源的`AccessLogSettings`属性。

`Auth`

配置授权，以控制对 API Gateway HTTP API 的访问。

有关更多信息，请参阅《API Gateway 开发者指南》中的 “ APIs 使用 JWT 授权者控制 HTTP 访问权限”。

类型：HttpApiAuth

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`CorsConfiguration`

管理所有 API Gateway HTTP 的跨源资源共享 (CORS)。 APIs以字符串形式指定要允许的域，或者指定一个 `HttpApiCorsConfiguration` 对象。请注意，CORS Amazon SAM 需要修改你的 OpenAPI 定义，因此 CORS 只有在指定`DefinitionBody`了该属性时才起作用。

有关更多信息，请参阅《API Gateway 开发人员指南》中的为 HTTP API 配置 CORS。

> **Note**
>
> 如果`CorsConfiguration`在 OpenAPI 定义和属性级别都进行了设置，则会将两个配置源 Amazon SAM 合并，且属性优先。如果将此属性设置为 `true`，则允许所有来源。

类型：字符串 | HttpApiCorsConfiguration

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`DefaultRouteSettings`

此 HTTP API 的默认路由设置。这些设置适用于所有路由，除非被某些路由的 `RouteSettings` 属性覆盖。

类型：RouteSettings

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::ApiGatewayV2::Stage`资源的`RouteSettings`属性。

`DefinitionBody`

描述您的 HTTP API 的 OpenAPI 定义。如果您未指定 a `DefinitionUri` 或 a`DefinitionBody`，则会根据您的模板配置`DefinitionBody`为您 Amazon SAM 生成。

类型：JSON

必需：否

Amazon CloudFormation 兼容性：此属性类似于`AWS::ApiGatewayV2::Api`资源的`Body`属性。如果提供了某些属性，则 Amazon SAM 可以在将内容传递给`DefinitionBody`之前将其插入或修改 Amazon CloudFormation。属性包括相应`EventSource` HttpApi `AWS::Serverless::Function`资源的类型`Auth`和类型。

`DefinitionUri`

定义 HTTP API 的 OpenAPI 定义的 Amazon Simple Storage Service (Amazon S3) URI、本地文件路径或位置对象。此属性引用的 Amazon S3 对象必须是有效的 OpenAPI 定义文件。如果您未指定`DefinitionUri`或`DefinitionBody`已指定，则会根据您的模板配置`DefinitionBody`为您 Amazon SAM 生成。

如果您提供本地文件路径，则模板必须经过包含 `sam deploy` 或 `sam package` 命令的工作流程，才能使定义正确转换。

您引用的外部 OpenApi 定义文件不支持内部函数。`DefinitionUri`要将 OpenApi 定义导入到模板中，请使用带有 Include 转换的`DefinitionBody`属性。

类型：字符串 | HttpApiDefinition

必需：否

Amazon CloudFormation 兼容性：此属性类似于`AWS::ApiGatewayV2::Api`资源的`BodyS3Location`属性。嵌套的 Amazon S3 属性的命名有所不同。

`Description`

HTTP API 资源的描述。

当您指定时`Description`， Amazon SAM 将通过设置`description`字段来修改 HTTP API 资源的OpenApi 定义。以下情况将导致错误：

- 该`DefinitionBody`属性是使用 Open API 定义中设置的`description`字段指定的，这会导致 Amazon SAM 无法解决的`description`字段冲突。
- 该`DefinitionUri`属性已指定 — Amazon SAM 不会修改从 Amazon S3 检索到的开放 API 定义。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`DisableExecuteApiEndpoint`

指定客户端是否可以使用默认 `execute-api` 端点 `https://{api_id}.execute-api.{region}.amazonaws.com` 调用您的 HTTP API。默认情况下，客户端可以使用默认端点调用您的 API。如果要求客户端仅使用自定义域名调用 API，请禁用默认端点。

要使用此属性，您必须指定 `DefinitionBody` 属性而不是 `DefinitionUri` 属性，或在 OpenAPI 定义中使用 `disableExecuteApiEndpoint` 定义 `x-amazon-apigateway-endpoint-configuration`。

类型：布尔值

必需：否

Amazon CloudFormation 兼容性：此属性类似于`AWS::ApiGatewayV2::Api`资源的 `DisableExecuteApiEndpoint`属性。它直接传递给 `x-amazon-apigateway-endpoint-configuration` 扩展的 `disableExecuteApiEndpoint` 属性，然后添加到 `AWS::ApiGatewayV2::Api` 资源的 `Body` 属性中。

`Domain`

为此 API Gateway HTTP API 配置自定义域。

类型：HttpApiDomainConfiguration

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`FailOnWarnings`

指定在遇到警告时是回滚 (`true`) 还是不回滚 (`false`) HTTP API 创建。默认值为 `false`。

类型：布尔值

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::ApiGatewayV2::Api`资源的`FailOnWarnings`属性。

`Name`

HTTP API 资源的名称。

当您指定时`Name`， Amazon SAM 将通过设置该字段来修改 HTTP API 资源的 OpenAPI 定义。`title`以下情况将导致错误：

- 该`DefinitionBody`属性是使用 Open API 定义中设置的`title`字段指定的，这会导致 Amazon SAM 无法解决的`title`字段冲突。
- 该`DefinitionUri`属性已指定 — Amazon SAM 不会修改从 Amazon S3 检索到的开放 API 定义。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`PropagateTags`

指明是否将 `Tags` 属性中的标签传递给 AWS::Serverless::HttpApi 生成的资源。指定 `True` 以在生成的资源中传播标签。

类型：布尔值

必需：否

默认值：`False`

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`RouteSettings`

此 HTTP API 的每个路由的路由设置。有关更多信息，请参阅 API Gateway 开发者指南 APIs中的使用 HTTP 路由。

类型：RouteSettings

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::ApiGatewayV2::Stage`资源的`RouteSettings`属性。

`StageName`

API 阶段的名称。如果未指定名称，则 Amazon SAM 使用 API Gateway 中的`$default`阶段。

类型：字符串

必需：否

默认值：$default

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::ApiGatewayV2::Stage`资源的`StageName`属性。

`StageVariables`

一个定义阶段变量的映射。变量名可以包含字母数字和下划线字符。这些值必须匹配 [A-Za-z0-9-._~:/?#&=,]+。

类型：Json

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::ApiGatewayV2::Stage`资源的`StageVariables`属性。

`Tags`

指定要添加到此 API Gateway 阶段的标签的映射（字符串到字符串）。密钥的长度可以在 1 到 128 个 Unicode 字符之间，并且不能包含前缀 `aws:`。您可以使用以下任一字符：Unicode 字母、数字、空格、_、 .、/、=、+ 和 - 的组合。值的长度可以在 1 到 256 个 Unicode 字符之间。

类型：映射

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

其他说明：该`Tags`属性 Amazon SAM 需要修改您的 OpenAPI 定义，因此只有在指定了该`DefinitionBody`属性时才会添加标签，如果指定了该属性，则不会添加任何标签。`DefinitionUri` Amazon SAM 自动添加`httpapi:createdBy:SAM`标签。标签还会添加到 `AWS::ApiGatewayV2::Stage` 资源和 `AWS::ApiGatewayV2::DomainName` 资源（如果已指定 `DomainName`）。

返回值

Ref

在将此资源的逻辑 ID 传递给内置 `Ref` 函数时，`Ref` 会返回底层 `AWS::ApiGatewayV2::Api` 资源的 API ID，例如 `a1bcdef2gh`。

有关使用 `Ref` 函数的更多信息，请参阅《Amazon CloudFormation 用户指南》中的 `Ref`。

示例

简单 HttpApi

以下示例显示了设置由 Lambda 函数支持的 HTTP API 端点所需的最低要求。此示例使用 Amazon SAM 创建的默认 HTTP API。

YAML

```
AWSTemplateFormatVersion: '2010-09-09'
Description: Amazon SAM template with a simple API definition
Resources:
  ApiFunction:
    Type: AWS::Serverless::Function
    Properties:
      Events:
        ApiEvent:
          Type: HttpApi
      Handler: index.handler
      InlineCode: |
        def handler(event, context):
            return {'body': 'Hello World!', 'statusCode': 200}
      Runtime: python3.7
Transform: AWS::Serverless-2016-10-31
```

HttpApi 使用身份验证

以下示例说明如何在 HTTP API 端点上设置授权。

YAML

```
Properties:
  FailOnWarnings: true
  Auth:
    DefaultAuthorizer: OAuth2
    Authorizers:
      OAuth2:
        AuthorizationScopes:
          - scope4
        JwtConfiguration:
          issuer: "https://www.example.com/v1/connect/oauth2"
          audience:
            - MyApi
        IdentitySource: "$request.querystring.param"
```

HttpApi使用 OpenAPI 定义

以下示例说明如何将 OpenAPI 定义添加到模板。

请注意，对于引用此 HTTP API HttpApi 的事件，请 Amazon SAM 填写所有缺失的 Lambda 集成。Amazon SAM 还会添加 HttpApi 事件引用的任何缺失路径。

YAML

```
Properties:
  FailOnWarnings: true
  DefinitionBody:
    info:
      version: '1.0'
      title:
        Ref: AWS::StackName
    paths:
      "/":
        get:
          security:
          - OpenIdAuth:
            - scope1
            - scope2
```

```
          responses: {}
    openapi: 3.0.1
    securitySchemes:
      OpenIdAuth:
        type: openIdConnect
        x-amazon-apigateway-authorizer:
          identitySource: "$request.querystring.param"
          type: jwt
          jwtConfiguration:
            audience:
            - MyApi
            issuer: https://www.example.com/v1/connect/oidc
          openIdConnectUrl: https://www.example.com/v1/connect/oidc/.well-known/openid-
configuration
```

HttpApi 使用配置设置

以下示例说明如何将 HTTP API 和阶段配置添加到模板。

YAML

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Parameters:
  StageName:
    Type: String
    Default: Prod

Resources:
  HttpApiFunction:
    Type: AWS::Serverless::Function
    Properties:
      InlineCode: |
          def handler(event, context):
              import json
              return {
                  "statusCode": 200,
                  "body": json.dumps(event),
              }
      Handler: index.handler
      Runtime: python3.7
      Events:
        ExplicitApi: # warning: creates a public endpoint
          Type: HttpApi
```

```
          Properties:
            ApiId: !Ref HttpApi
            Method: GET
            Path: /path
            TimeoutInMillis: 15000
            PayloadFormatVersion: "2.0"
            RouteSettings:
              ThrottlingBurstLimit: 600

  HttpApi:
    Type: AWS::Serverless::HttpApi
    Properties:
      StageName: !Ref StageName
      Tags:
        Tag: Value
      AccessLogSettings:
        DestinationArn: !GetAtt AccessLogs.Arn
        Format: $context.requestId
      DefaultRouteSettings:
        ThrottlingBurstLimit: 200
      RouteSettings:
        "GET /path":
          ThrottlingBurstLimit: 500 # overridden in HttpApi Event
      StageVariables:
        StageVar: Value
      FailOnWarnings: true

  AccessLogs:
    Type: AWS::Logs::LogGroup

Outputs:
  HttpApiUrl:
    Description: URL of your API endpoint
    Value:
      Fn::Sub: 'https://${HttpApi}.execute-api.${AWS::Region}.${AWS::URLSuffix}/
${StageName}/'
  HttpApiId:
    Description: Api id of HttpApi
    Value:
      Ref: HttpApi
```

HttpApiAuth

配置授权，以控制对 Amazon API Gateway HTTP API 的访问。

有关配置 HTTP 访问权限的更多信息 APIs，请参阅《API Gateway 开发者指南》中的 API Gateway 中控制和管理对 HTTP API 的访问权限。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
Authorizers: OAuth2Authorizer | LambdaAuthorizer
DefaultAuthorizer: String
EnableIamAuthorizer: Boolean
```

属性

`Authorizers`

用于控制对 API Gateway API 的访问的授权方。

类型：OAuth2授权者 | LambdaAuthorizer

必需：否

默认值：无

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

其他说明： Amazon SAM 将授权者添加到 OpenAPI 定义中。

`DefaultAuthorizer`

指定默认授权方，以用于授权对 API Gateway API 的 API 调用。如果 `EnableIamAuthorizer` 设置为 `true`，您可以指定 `AWS_IAM` 作为默认授权方。否则，请指定您在 `Authorizers` 中定义的授权方。

类型：字符串

必需：否

默认值：无

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`EnableIamAuthorizer`

指定是否要对 API 路由使用 IAM 授权。

类型：布尔值

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

示例

OAuth 2.0 授权者

OAuth 2.0 授权方示例

YAML

```
Auth:
  Authorizers:
    OAuth2Authorizer:
      AuthorizationScopes:
        - scope1
        - scope2
      JwtConfiguration:
        issuer: "https://www.example.com/v1/connect/oauth2"
        audience:
          - MyApi
      IdentitySource: "$request.querystring.param"
  DefaultAuthorizer: OAuth2Authorizer
```

IAM 授权方

IAM 授权方示例

YAML

```
Auth:
  EnableIamAuthorizer: true
  DefaultAuthorizer: AWS_IAM
```

LambdaAuthorizer

配置 Lambda 授权机构，使用函数控制对您的 Amazon API Gateway HTTP API 的访问。 Amazon Lambda

有关更多信息和示例，请参阅《API Gateway 开发者指南》 APIs中的 “使用 HTTP 的 Amazon Lambda 授权方”。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
AuthorizerPayloadFormatVersion: String
EnableFunctionDefaultPermissions: Boolean
EnableSimpleResponses: Boolean
FunctionArn: String
FunctionInvokeRole: String
Identity: LambdaAuthorizationIdentity
```

属性

`AuthorizerPayloadFormatVersion`

指定发送到 HTTP API Lambda 授权方的负载的格式。对于 HTTP API Lambda 授权方必须指定。

这会传递到 OpenAPI 定义的 `securitySchemes` 部分中 `x-amazon-apigateway-authorizer` 的 `authorizerPayloadFormatVersion` 部分。

有效值：`1.0` 或 `2.0`

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`EnableFunctionDefaultPermissions`

默认情况下，未授予 HTTP API 资源调用 Lambda 授权方的权限。将此属性指定为 `true` 以自动在您的 HTTP API 资源和 Lambda 授权方之间创建权限。

类型：布尔值

必需：否

默认值：`false`

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`EnableSimpleResponses`

指定 Lambda 授权方是否以简单格式返回响应。默认情况下，Lambda 授权机构必须返回 Amazon Identity and Access Management (IAM) 策略。如果启用，Lambda 授权方可以返回布尔值，而不是 IAM 策略。

这会传递到 OpenAPI 定义的 `securitySchemes` 部分中 `x-amazon-apigateway-authorizer` 的 `enableSimpleResponses` 部分。

类型：布尔值

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`FunctionArn`

为 API 提供授权的 Lambda 函数的 Amazon 资源名称（ARN）。

这会传递到 OpenAPI 定义的 `securitySchemes` 部分中 `x-amazon-apigateway-authorizer` 的 `authorizerUri` 部分。

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`FunctionInvokeRole`

具有 API Gateway 调用授权方函数所需凭证的 IAM 角色的 ARN。如果函数的基于资源的策略未授予 API Gateway `lambda:InvokeFunction` 权限，请指定此参数。

这会传递到 OpenAPI 定义的 `securitySchemes` 部分中 `x-amazon-apigateway-authorizer` 的 `authorizerCredentials` 部分。

有关更多信息，请参阅《API Gateway 开发人员指南》中的创建 Lambda 授权方。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`Identity`

在授权方传入请求中指定 `IdentitySource`。

这会传递到 OpenAPI 定义的 `securitySchemes` 部分中 `x-amazon-apigateway-authorizer` 的 `identitySource` 部分。

类型：LambdaAuthorizationIdentity

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

示例

LambdaAuthorizer

LambdaAuthorizer 示例

YAML

```
Auth:
  Authorizers:
    MyLambdaAuthorizer:
      AuthorizerPayloadFormatVersion: 2.0
      FunctionArn:
        Fn::GetAtt:
          - MyAuthFunction
          - Arn
```

```
      FunctionInvokeRole:
        Fn::GetAtt:
          - LambdaAuthInvokeRole
          - Arn
      Identity:
        Headers:
          - Authorization
```

LambdaAuthorizationIdentity

Use 属性可用于在 Lambda 授权方的传入请求 IdentitySource 中指定。有关身份源的更多信息，请参阅《API Gateway 开发人员指南》中的身份源。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
Context: List
Headers: List
QueryStrings: List
ReauthorizeEvery: Integer
StageVariables: List
```

属性

`Context`

将给定的上下文字符串转换为映射表达式列表，格式为 `$context.contextString`。

类型：列表

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`Headers`

将标头转换为映射表达式列表，格式为 `$request.header.name`。

类型：列表

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`QueryStrings`

将给定的查询字符串转换为映射表达式列表，格式为 `$request.querystring.queryString`。

类型：列表

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`ReauthorizeEvery`

time-to-live(TTL) 周期，以秒为单位，用于指定 API Gateway 缓存授权方结果的时长。如果您指定一个大于 0 的值，API Gateway 将缓存授权方响应。最大值为 3600 秒（1 小时）。

类型：整数

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`StageVariables`

将给定的阶段变量转换为映射表达式列表，格式为 `$stageVariables.stageVariable`。

类型：列表

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

示例

LambdaRequestIdentity

Lambda 请求身份示例

YAML

```
Identity:
  QueryStrings:
    - auth
  Headers:
    - Authorization
  StageVariables:
    - VARIABLE
  Context:
    - authcontext
  ReauthorizeEvery: 100
```

OAuth2Authorizer

OAuth 2.0 授权方（也称为 JSON 网络令牌 (JWT) 授权方）的定义。

有关更多信息，请参阅《API Gateway 开发者指南》中的 “ APIs 使用 JWT 授权者控制 HTTP 访问权限”。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
  AuthorizationScopes: List
  IdentitySource: String
  JwtConfiguration: Map
```

属性

`AuthorizationScopes`

此授权方的授权范围列表。

类型：列表

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`IdentitySource`

此授权方的身份源表达式。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`JwtConfiguration`

此授权方的 JWT 配置。

这会传递到 OpenAPI 定义的 `securitySchemes` 部分中 `x-amazon-apigateway-authorizer` 的 `jwtConfiguration` 部分。

> **Note**
>
> 属`issuer`性和`audience`不区分大小写，可以使用 OpenAPI 中的小写字母，也可以使用大写字母`Issuer`和。`Audience` AWS::ApiGatewayV2::Authorizer

类型：映射

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

示例

OAuth 2.0 授权者

OAuth 2.0 授权方示例

YAML

```
Auth:
  Authorizers:
    OAuth2Authorizer:
```

```
      AuthorizationScopes:
        - scope1
      JwtConfiguration:
        issuer: "https://www.example.com/v1/connect/oauth2"
        audience:
          - MyApi
      IdentitySource: "$request.querystring.param"
  DefaultAuthorizer: OAuth2Authorizer
```

HttpApiCorsConfiguration

管理您的 HTTP 的跨源资源共享 (CORS)。 APIs使用字符串指定允许使用的域，或使用其他 Cors 配置指定词典。注意：Cors 要求 SAM 修改你的 OpenAPI 定义，因此它仅适用于属性中`DefinitionBody`定义的 OpenApi 内联定义。

有关 CORS 的更多信息，请参阅《API Gateway 开发人员指南》中的为 HTTP API 配置 CORS。

注意：如果 HttpApiCorsConfiguration 在 OpenAPI 和属性级别同时设置，则会将它们 Amazon SAM 合并，且属性优先。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
AllowCredentials: Boolean
AllowHeaders: List
AllowMethods: List
AllowOrigins: List
ExposeHeaders: List
MaxAge: Integer
```

属性

`AllowCredentials`

指定是否在 CORS 请求中包含凭证。

类型：布尔值

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`AllowHeaders`

表示允许的标头集合。

类型：列表

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`AllowMethods`

表示允许的 HTTP 方法集合。

类型：列表

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`AllowOrigins`

表示允许的源集合。

类型：列表

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`ExposeHeaders`

表示公开的标头集合。

类型：列表

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`MaxAge`

浏览器应缓存预检请求结果的秒数。

类型：整数

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

示例

HttpApiCorsConfiguration

HTTP API Cors 配置示例。

YAML

```
CorsConfiguration:
  AllowOrigins:
    - "https://example.com"
  AllowHeaders:
    - x-apigateway-header
  AllowMethods:
    - GET
  MaxAge: 600
  AllowCredentials: true
```

HttpApiDefinition

定义 API 的 OpenAPI 文档。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
  Bucket: String
  Key: String
  Version: String
```

属性

Bucket

存储了 OpenAPI 文件的 Amazon S3 桶的名称。

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::ApiGatewayV2::ApiBodyS3Location`数据类型的`Bucket`属性。

Key

OpenAPI 文件的 Amazon S3 密钥。

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::ApiGatewayV2::ApiBodyS3Location`数据类型的`Key`属性。

Version

对于受版本控制的对象，是 OpenAPI 文件的版本。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::ApiGatewayV2::ApiBodyS3Location`数据类型的`Version`属性。

示例

定义 Uri 示例

API 定义示例

YAML

```
DefinitionUri:
  Bucket: sam-s3-demo-bucket-name
```

```
  Key: mykey-name
  Version: 121212
```

HttpApiDomainConfiguration

为 API 配置自定义域。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
  BasePath: List
  CertificateArn: String
  DomainName: String
  EndpointConfiguration: String
  MutualTlsAuthentication: MutualTlsAuthentication
  OwnershipVerificationCertificateArn: String
  Route53: Route53Configuration
  SecurityPolicy: String
```

属性

`BasePath`

要使用 Amazon API Gateway 域名配置的基本路径列表。

类型：列表

必需：否

默认值：/

Amazon CloudFormation 兼容性：此属性类似于`AWS::ApiGatewayV2::ApiMapping`资源的`ApiMappingKey`属性。 Amazon SAM 创建多个`AWS::ApiGatewayV2::ApiMapping`资源，每个资源在此属性中指定的值一个。

`CertificateArn`

该域名终端节点的 Amazon 托管证书的 Amazon 资源名称 (ARN)。 Amazon Certificate Manager 是唯一支持的来源。

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::ApiGateway2::DomainName` `DomainNameConfiguration`资源的`CertificateArn`属性。

`DomainName`

API Gateway API 的自定义域名。不支持大写字母。

Amazon SAM 设置此属性时会生成`AWS::ApiGatewayV2::DomainName`资源。有关此场景的更多信息，请参阅DomainName属性已指定。有关生成的 Amazon CloudFormation 资源的信息，请参阅生成的 Amazon CloudFormation 资源用于 Amazon SAM。

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::ApiGateway2::DomainName`资源的`DomainName`属性。

`EndpointConfiguration`

定义要映射到自定义域的 API Gateway 端点的类型。此属性的值决定了该`CertificateArn`属性的映射方式 Amazon CloudFormation。

HTTP 的唯一有效值 APIs 是`REGIONAL`。

类型：字符串

必需：否

默认值：`REGIONAL`

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`MutualTlsAuthentication`

自定义域名的相互传输层安全性协议（TLS）身份验证配置。

类型：MutualTlsAuthentication

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::ApiGatewayV2::DomainName`资源的`MutualTlsAuthentication`属性。

`OwnershipVerificationCertificateArn`

ACM 颁发的用于验证自定义域所有权的公有证书的 ARN。只有在配置双向 TLS 并且为 `CertificateArn` 指定了 ACM 导入的或私有 CA 证书 ARN 时才需要。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::ApiGatewayV2::DomainNameDomainNameConfiguration`数据类型的`OwnershipVerificationCertificateArn`属性。

`Route53`

定义 Amazon Route 53 配置。

类型：Route53Configuration

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`SecurityPolicy`

此域名的安全策略的 TLS 版本。

HTTP 的唯一有效值 APIs 是`TLS_1_2`。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::ApiGatewayV2::DomainNameDomainNameConfiguration`数据类型的`SecurityPolicy`属性。

示例

DomainName

DomainName 示例

YAML

```
Domain:
  DomainName: www.example.com
  CertificateArn: arn-example
  EndpointConfiguration: REGIONAL
  Route53:
    HostedZoneId: Z1PA6795UKMFR9
  BasePath:
    - foo
    - bar
```

Route53Configuration

为 API 配置 Route53 记录集。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
  DistributionDomainName: String
  EvaluateTargetHealth: Boolean
  HostedZoneId: String
  HostedZoneName: String
  IpV6: Boolean
  Region: String
  SetIdentifier: String
```

属性

`DistributionDomainName`

配置 API 自定义域名的自定义分配。

类型：字符串

必需：否

默认：使用 API Gateway 分配。

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Route53::RecordSetGroup` `AliasTarget`资源的`DNSName`属性。

其他说明：CloudFront分配的域名。

`EvaluateTargetHealth`

如果 EvaluateTargetHealth 为 true，则别名记录将继承引用 Amazon 资源的运行状况，例如 Elastic Load Balancing 负载均衡器或托管区域中的其他记录。

类型：布尔值

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Route53::RecordSetGroup AliasTarget`资源的`EvaluateTargetHealth`属性。

其他说明：当别名目标为 CloudFront 分布时，您不能 EvaluateTargetHealth 将其设置为 true。

`HostedZoneId`

要在其中创建记录的托管区的 ID。

指定 `HostedZoneName` 或 `HostedZoneId`，但不能同时指定两者。如果您拥有多个使用相同域名的托管区域，则必须使用 `HostedZoneId` 指定托管区。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Route53::RecordSetGroup RecordSet`资源的`HostedZoneId`属性。

`HostedZoneName`

要在其中创建记录的托管区的名称。您必须包含结尾圆点（例如 `www.example.com.`）作为 `HostedZoneName` 的一部分。

指定 `HostedZoneName` 或 `HostedZoneId`，但不能同时指定两者。如果您拥有多个使用相同域名的托管区域，则必须使用 `HostedZoneId` 指定托管区。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Route53::RecordSetGroup RecordSet`资源的`HostedZoneName`属性。

`IpV6`

设置此属性后，将 Amazon SAM 创建一个`AWS::Route53::RecordSet`资源并将提供的资源的 Type 设置`AAAA`为 HostedZone。

类型：布尔值

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`Region`

仅限基于延迟的资源记录集：您创建此资源记录集所引用的资源的亚马逊 EC2 区域。资源通常是 Amazon 资源，例如 EC2 实例或 ELB 负载均衡器，并由 IP 地址或 DNS 域名引用，具体取决于记录类型。

当 Amazon Route 53 收到针对您已为其创建延迟资源记录集的域名和类型的 DNS 查询时，Route 53 会选择最终用户与关联亚马逊 EC2 地区之间延迟最低的延迟资源记录集。然后，Route 53 会返回与所选资源记录集相关的值。

请注意以下几点：

- 您只能为每个延迟资源记录集指定一个 `ResourceRecord`。
- 您只能为每个 Amazon EC2 地区创建一个延迟资源记录集。
- 您无需为所有 Amazon EC2 区域创建延迟资源记录集。Route 53 会从您已创建延迟资源记录集的区域中选择延迟性能最佳的区域。
- 您不能创建 `Name` 和 `Type` 元素的值与延迟资源记录集相同的非延迟资源记录集。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Route53::RecordSetGroupRecordSet`数据类型的 `Region`属性。

`SetIdentifier`

具有简单策略以外的路由策略的资源记录集：用于区分具有相同名称和类型组合的多个资源记录集的标识符，如名为 acme.example.com 且类型为 A 的多个加权资源记录集。在一组具有相同名称和类型的资源记录集中，每个资源记录集的 `SetIdentifier` 值必须唯一。

有关路由策略的信息，请参阅《Amazon Route 53 开发人员指南》中的选择路由策略。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Route53::RecordSetGroupRecordSet`数据类型的 `SetIdentifier`属性。

示例

Route 53 配置示例

此示例演示了如何配置 Route 53。

YAML

```
Domain:
  DomainName: www.example.com
  CertificateArn: arn-example
  EndpointConfiguration: EDGE
  Route53:
    HostedZoneId: Z1PA6795UKMFR9
    EvaluateTargetHealth: true
    DistributionDomainName: xyz
```

AWS::Serverless::LayerVersion

创建包含 Lambda 函数 LayerVersion 所需的库或运行时代码的 Lambda。

该AWS::Serverless::LayerVersion资源还支持 res `Metadata` ource 属性，因此您可以指示 Amazon SAM 构建应用程序中包含的图层。有关构建层的更多信息，请参阅 在中构建 Lambda 图层 Amazon SAM。

重要说明：自从UpdateReplacePolicy资源属性发布以来 Amazon CloudFormation，AWS::Lambda::LayerVersion（推荐）提供的好处与相同AWS::Serverless::LayerVersion。

转换无服务器 LayerVersion 时，SAM 还会转换资源的逻辑 ID，这样资源更新 CloudFormation 时就不会自动删除旧 LayerVersions 的 ID。

> **Note**
>
> 部署到时 Amazon CloudFormation， Amazon SAM 会将您的 Amazon SAM 资源转换为 Amazon CloudFormation 资源。有关更多信息，请参阅 生成的 Amazon CloudFormation 资源用于 Amazon SAM。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
Type: AWS::Serverless::LayerVersion
Properties:
  CompatibleArchitectures: List
  CompatibleRuntimes: List
  ContentUri: String | LayerContent
  Description: String
  LayerName: String
  LicenseInfo: String
  PublishLambdaVersion: Boolean
  RetentionPolicy: String
```

属性

`CompatibleArchitectures`

为层版本指定支持的指令集架构。

有关此属性更多信息，请参阅《Amazon Lambda 开发人员指南》中的 Lambda 指令集架构。

有效值：`x86_64`、`arm64`

类型：列表

必需：否

默认值：`x86_64`

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::LayerVersion`资源的`CompatibleArchitectures`属性。

`CompatibleRuntimes`

与此兼容的运行时列表。 LayerVersion

类型：列表

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::LayerVersion`资源的`CompatibleRuntimes`属性。

`ContentUri`

Amazon S3 Uri、本地文件夹的路径或层代码的 LayerContent 对象。

如果提供了 Amazon S3 URI 或 LayerContent 对象，则引用的亚马逊 S3 对象必须是包含 Lambda 层内容的有效 ZIP 档案。

如果提供了本地文件夹的路径，则为使内容正确转换，模板必须经过包括 sam build 以及后续的 sam deploy 或 sam package 的工作流程。默认情况下，相对路径是根据 Amazon SAM 模板的位置进行解析的。

类型：字符串 | LayerContent

必需：是

Amazon CloudFormation 兼容性：此属性类似于`AWS::Lambda::LayerVersion`资源的`Content`属性。嵌套的 Amazon S3 属性的命名有所不同。

`Description`

该层的描述。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::LayerVersion`资源的`Description`属性。

`LayerName`

层的名称或 Amazon 资源名称（ARN）。

类型：字符串

必需：否

默认：资源逻辑 ID

Amazon CloudFormation 兼容性：此属性类似于`AWS::Lambda::LayerVersion`资源的`LayerName`属性。如果您没有指定名称，则将使用资源的逻辑 ID 作为名称。

`LicenseInfo`

有关此许可证的信息 LayerVersion。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Lambda::LayerVersion`资源的`LicenseInfo`属性。

`PublishLambdaVersion`

一个 opt-in 属性，每当引用的资源发生变化时，它都会创建一个新的 Lambda 版本。`LayerVersion`当使用`AutoPublishAlias`和`AutoPublishAliasAllProperties`在连接的 Lambda 函数中启用时，将为对资源所做的每一次更改创建一个新的 Lambda 版本。`LayerVersion`

类型：布尔值

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`RetentionPolicy`

此属性指定当您删除资源时，是保留还是删除 `LayerVersion` 的旧版本。如果您在更新或替换资源时需要保留 `LayerVersion` 的旧版本，则必须启用 `UpdateReplacePolicy` 属性。有关执行此操作的信息，请参阅《Amazon CloudFormation 用户指南》中的 `UpdateReplacePolicy` 属性。

有效值：`Retain` 或 `Delete`

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

其他注意事项：指定后`Retain`， Amazon SAM 会`DeletionPolicy: Retain`向转换后的`AWS::Lambda::LayerVersion`资源添加一个支持的资源属性 Amazon SAM。

返回值

Ref

当向`Ref`内部函数提供此资源的逻辑 ID 时，它将返回底层 Lambda 的资源 ARN。 LayerVersion

有关使用 `Ref` 函数的更多信息，请参阅《Amazon CloudFormation 用户指南》中的 `Ref`。

示例

LayerVersionExample

的示例 LayerVersion

YAML

```
Properties:
  LayerName: MyLayer
  Description: Layer description
  ContentUri: 's3://sam-s3-demo-bucket/my-layer.zip'
  CompatibleRuntimes:
    - nodejs10.x
    - nodejs12.x
  LicenseInfo: 'Available under the MIT-0 license.'
  RetentionPolicy: Retain
```

LayerContent

一个 ZIP 存档，其中包含 Lambda 层的内容。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
  Bucket: String
```

```
  Key: String
  Version: String
```

属性

Bucket

层存档的 Amazon S3 桶。

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性直接传递给AWS::Lambda::LayerVersionContent数据类型的S3Bucket属性。

Key

层存档的 Amazon S3 密钥。

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性直接传递给AWS::Lambda::LayerVersionContent数据类型的S3Key属性。

Version

对于进行版本控制的对象，则为要使用的层存档对象版本。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给AWS::Lambda::LayerVersionContent数据类型的S3ObjectVersion属性。

示例

LayerContent

层内容示例

YAML

```
LayerContent:
  Bucket: amzn-s3-demo-bucket-name
  Key: mykey-name
  Version: 121212
```

AWS::Serverless::SimpleTable

创建具有单个属性主键的 DynamoDB 表。当只需要通过主键访问数据时，它很有用。

要获得更高级的功能，请使用 AWS::DynamoDB::Table资源在 Amazon CloudFormation。这些资源可用于。 Amazon SAM它们非常全面，可提供进一步的定制，包括 key schema 和 resource policy定制。

> **Note**
>
> 部署到时 Amazon CloudFormation， Amazon SAM 会将您的 Amazon SAM 资源转换为 Amazon CloudFormation 资源。有关更多信息，请参阅 生成的 Amazon CloudFormation 资源用于 Amazon SAM。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
Type: AWS::Serverless::SimpleTable
Properties:
  PointInTimeRecoverySpecification: PointInTimeRecoverySpecification
  PrimaryKey: PrimaryKeyObject
  ProvisionedThroughput: ProvisionedThroughput
  SSESpecification: SSESpecification
  TableName: String
  Tags: Map
```

属性

`PointInTimeRecoverySpecification`

用于启用时间点恢复的设置。

类型:PointInTimeRecoverySpecification

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::DynamoDB::Table`资源的`PointInTimeRecoverySpecification`属性。

`PrimaryKey`

用作表主键的属性名称和类型。如果未提供，则主键将为 `String`，值为 `id`。

> **Note**
> 创建此资源后，无法修改此属性的值。

类型：PrimaryKeyObject

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`ProvisionedThroughput`

读取和写入吞吐量配置信息。

如果未指定 `ProvisionedThroughput`，则将 `BillingMode` 指定为 `PAY_PER_REQUEST`。

类型：ProvisionedThroughput

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::DynamoDB::Table`资源的`ProvisionedThroughput`属性。

`SSESpecification`

指定用于启用服务器端加密的设置。

类型：SSESpecification

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::DynamoDB::Table`资源的`SSESpecification`属性。

`TableName`

DynamoDB 表的名称。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::DynamoDB::Table`资源的`TableName`属性。

`Tags`

一个地图（字符串到字符串），用于指定要添加到其中的标签 SimpleTable。有关标签的有效键和值的详细信息，请参阅《Amazon CloudFormation 用户指南》中的资源标签。

类型：映射

必需：否

Amazon CloudFormation 兼容性：此属性类似于`AWS::DynamoDB::Table`资源的`Tags`属性。SAM 中的 Tags 属性由 Key: Value CloudFormation 对组成；其中包含标签对象的列表。

返回值

Ref

当该资源的逻辑 ID 提供给 Ref 内置函数时，它将返回底层 DynamoDB 表的资源名称。

有关使用 `Ref` 函数的更多信息，请参阅《Amazon CloudFormation 用户指南》中的 `Ref`。

示例

SimpleTableExample

的示例 SimpleTable

YAML

```
Properties:
```

```
  TableName: my-table
  Tags:
    Department: Engineering
    AppType: Serverless
```

PrimaryKeyObject

描述主键属性的对象。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
  Name: String
  Type: String
```

属性

Name

主键的属性名称。

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::DynamoDB::TableAttributeDefinition`数据类型的`AttributeName`属性。

其他说明：此属性也传递给`AWS::DynamoDB::Table KeySchema`数据类型的AttributeName属性。

Type

主键的数据类型。

有效值：`String`、`Number`、`Binary`

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::DynamoDB::TableAttributeDefinition`数据类型的`AttributeType`属性。

示例

PrimaryKey

主键示例。

YAML

```
Properties:
  PrimaryKey:
    Name: MyPrimaryKey
    Type: String
```

AWS::Serverless::StateMachine

创建 Amazon Step Functions 状态机，您可以使用它来编排 Amazon Lambda 函数和其他 Amazon 资源，以形成复杂而强大的工作流程。

有关 Step Functions 的更多信息，请参阅《Amazon Step Functions 开发人员指南》。

> **Note**
> 部署到时 Amazon CloudFormation， Amazon SAM 会将您的 Amazon SAM 资源转换为 Amazon CloudFormation 资源。有关更多信息，请参阅 生成的 Amazon CloudFormation 资源用于 Amazon SAM。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
Type: AWS::Serverless::StateMachine
Properties:
  AutoPublishAlias: String
  UseAliasAsEventTarget: Boolean
  Definition: Map
```

```
  DefinitionSubstitutions: Map
  DefinitionUri: String | S3Location
  DeploymentPreference: DeploymentPreference
  Events: EventSource
  Logging: LoggingConfiguration
  Name: String
  PermissionsBoundary: String
  Policies: String | List | Map
  PropagateTags: Boolean
  RolePath: String
  Role: String
  Tags: Map
  Tracing: TracingConfiguration
  Type: String
```

属性

AutoPublishAlias

状态机别名的名称。要详细了解如何使用 Step Functions 状态机别名，请参阅《Amazon Step Functions 开发人员指南》中的使用版本与别名功能管理持续部署。

使用 DeploymentPreference 为别名配置部署首选项。如果您未指定DeploymentPreference，则 Amazon SAM 会将流量配置为一次性切换到较新的状态机版本。

Amazon SAM 默认情况下，将版本UpdateReplacePolicyRetain的DeletionPolicy和设置为。以前的版本不会被自动删除。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给AWS::StepFunctions::StateMachineAlias资源的 Name属性。

UseAliasAsEventTarget

指示是否将使用 AutoPublishAlias 属性创建的别名传递给使用事件定义的事件源目标。

指定 True 使用别名作为事件的目标。

类型：布尔值

必需：否

默认值：`False`

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`Definition`

状态机定义是一个对象，其中对象的格式与 Amazon SAM 模板文件的格式相匹配，例如 JSON 或 YAML。状态机定义遵循 Amazon States Language。

有关内联状态机定义的示例，请参见 示例。

您必须提供 `Definition` 或 `DefinitionUri`。

类型：映射

必需：条件

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`DefinitionSubstitutions`

在状态机定义中指定占位符变量的映射的 string-to-string地图。这使您能够将运行时获得的值（例如，从内置函数）注入到状态机定义中。

类型：映射

必需：否

Amazon CloudFormation 兼容性：此属性类似于`AWS::StepFunctions::StateMachine`资源的`DefinitionSubstitutions`属性。如果在内联状态机定义中指定了任何内部函数，则向该属性 Amazon SAM 添加条目以将其注入状态机定义中。

`DefinitionUri`

Amazon Simple Storage Service (Amazon S3) URI 或以 Amazon States Language 编写的状态机定义的本地文件路径。

如果您提供本地文件路径，则模板必须通过包含 `sam deploy` 或 `sam package` 命令的工作流程才能使定义正确转换。为此，必须使用 Amazon SAM CLI 版本 0.52.0 或更高版本。

您必须提供 `Definition` 或 `DefinitionUri`。

类型：字符串 | S3Location

必需：条件

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::StepFunctions::StateMachine`资源的`DefinitionS3Location`属性。

`DeploymentPreference`

启用和配置逐步状态机部署的设置。要详细了解 Step Functions 逐步部署，请参阅《Amazon Step Functions 开发人员指南》中的使用版本与别名功能管理持续部署。

在配置此属性之前指定 `AutoPublishAlias`。您的 `DeploymentPreference` 设置将应用于通过 `AutoPublishAlias` 指定的别名。

指定后`DeploymentPreference`， Amazon SAM 会自动生成`StateMachineVersionArn`子属性值。

类型：DeploymentPreference

必需：否

Amazon CloudFormation 兼容性： Amazon SAM 生成`StateMachineVersionArn`属性值并将其附加到资源的`DeploymentPreference`属性，`DeploymentPreference`然后传递`DeploymentPreference`给`AWS::StepFunctions::StateMachineAlias`资源的属性。

`Events`

指定触发此状态机的事件。事件由一个类型和一组依赖于该类型的属性组成。

类型：EventSource

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`Logging`

定义记录哪些执行历史事件以及它们的记录位置。

类型：LoggingConfiguration

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::StepFunctions::StateMachine`资源的`LoggingConfiguration`属性。

`Name`

状态机的名称。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::StepFunctions::StateMachine`资源的`StateMachineName`属性。

`PermissionsBoundary`

此状态机执行角色使用的权限边界的 ARN。此属性仅在为您生成角色时有效。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::IAM::Role`资源的`PermissionsBoundary`属性。

`Policies`

此状态机的权限策略。策略将附加到状态机的默认 Amazon Identity and Access Management (IAM) 执行角色中。

此属性接受单个值或值列表。允许的值包括：

- Amazon SAM策略模板.
- 这些区域有：ARN Amazon 托管策略或客户托管策略。
- 以下列表中 Amazon 托管策略的名称。
- 内联 IAM 策略的格式为 YAML 作为地图。

> **Note**
>
> 如果指定 `Role` 属性，则将忽略该属性。

类型：字符串 | 列表 | 映射

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`PropagateTags`

指明是否将 `Tags` 属性中的标签传递给 AWS::Serverless::StateMachine 生成的资源。指定 `True` 以在生成的资源中传播标签。

类型：布尔值

必需：否

默认值：`False`

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`Role`

用作此状态机执行角色的 IAM 角色的 ARN。

类型：字符串

必需：条件

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::StepFunctions::StateMachine`资源的 `RoleArn`属性。

`RolePath`

状态机的 IAM 执行角色的路径。

生成角色时请使用此属性。当使用 `Role` 属性指定角色时，请勿使用。

类型：字符串

必需：条件

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::IAM::Role`资源的`Path`属性。

`Tags`

指定添加到状态机的标签和相应的执行角色的 string-to-string地图。有关标签的有效键和值的信息，请参阅的 Tags 属性 AWS::StepFunctions::StateMachine资源。

类型：映射

必需：否

Amazon CloudFormation 兼容性：此属性类似于`AWS::StepFunctions::StateMachine`资源的`Tags`属性。 Amazon SAM 自动为该资源以及为其生成的默认角色添加`stateMachine:createdBy:SAM`标签。

`Tracing`

选择是否 Amazon X-Ray 为状态机启用。有关使用 X-Ray 和 Step Functions 的更多信息，请参阅《Amazon Step Functions 开发人员指南》中的 Amazon X-Ray 与 Step Functions。

类型：TracingConfiguration

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::StepFunctions::StateMachine`资源的`TracingConfiguration`属性。

`Type`

状态机的类型。

有效值：`STANDARD` 或 `EXPRESS`

类型：字符串

必需：否

默认值：`STANDARD`

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::StepFunctions::StateMachine`资源的`StateMachineType`属性。

返回值

Ref

在将此资源的逻辑 ID 提供给 Ref 内置函数时，Ref 会返回底层 `AWS::StepFunctions::StateMachine` 资源的 Amazon 资源名称（ARN）。

有关使用 `Ref` 函数的更多信息，请参阅《Amazon CloudFormation 用户指南》中的 `Ref`。

Fn:: GetAtt

`Fn::GetAtt` 返回一个此类型指定属性的值。以下为可用属性和示例返回值。

有关使用 `Fn::GetAtt` 的更多信息，请参阅《Amazon CloudFormation 用户指南》中的 `Fn::GetAtt`。

`Name`

返回状态机的名称，例如 `HelloWorld-StateMachine`。

示例

状态机定义文件

以下是内联状态机定义的示例，该定义允许 Lambda 函数调用状态机。请注意，此示例要求 `Role` 属性配置适当的策略以允许调用。`my_state_machine.asl.json` 文件必须以 Amazon States Language 编写。

在此示例中，这些`DefinitionSubstitution`条目允许状态机包含在 Amazon SAM 模板文件中声明的资源。

YAML

```
MySampleStateMachine:
  Type: AWS::Serverless::StateMachine
  Properties:
    DefinitionUri: statemachine/my_state_machine.asl.json
    Role: arn:aws:iam::123456123456:role/service-role/my-sample-role
    Tracing:
      Enabled: true
    DefinitionSubstitutions:
      MyFunctionArn: !GetAtt MyFunction.Arn
      MyDDBTable: !Ref TransactionTable
```

内联状态机定义

以下是内联状态机定义的示例。

在此示例中， Amazon SAM 模板文件是用 YAML 编写的，因此状态机定义也在 YAML 中。要在 JSON 中声明内联状态机定义，请使用 JSON 编写 Amazon SAM 模板文件。

YAML

```
MySampleStateMachine:
  Type: AWS::Serverless::StateMachine
  Properties:
    Definition:
      StartAt: MyLambdaState
      States:
        MyLambdaState:
          Type: Task
          Resource: arn:aws:lambda:us-east-1:123456123456:function:my-sample-lambda-app
          End: true
    Role: arn:aws:iam::123456123456:role/service-role/my-sample-role
    Tracing:
      Enabled: true
```

EventSource

描述触发状态机的事件源的对象。每个事件都由一个类型和一组依赖于该类型的属性组成。有关每个事件源的属性的更多信息，请参阅与具体类型对应的子主题。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
  Properties: Schedule | ScheduleV2 | CloudWatchEvent | EventBridgeRule | Api
  Type: String
```

属性

`Properties`

描述此事件映射的属性的对象。这组属性必须符合定义的 `Type`。

类型：日程安排 | ScheduleV2 | | | A CloudWatchEventpi EventBridgeRule

必需：是

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

Type

事件类型。

有效值：Api、Schedule、ScheduleV2、CloudWatchEvent、EventBridgeRule

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

示例

API

以下是 API 类型事件的示例。

YAML

```
ApiEvent:
  Type: Api
  Properties:
    Method: get
    Path: /group/{user}
    RestApiId:
      Ref: MyApi
```

Api

描述 Api 事件源类型的对象。如果定义了 AWS::Serverless::Api 资源，则路径和方法值必须与 API 的 OpenAPI 定义中的操作相对应。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
  Auth: ApiStateMachineAuth
```

```
  Method: String
  Path: String
  RestApiId: String
  UnescapeMappingTemplate: Boolean
```

属性

`Auth`

此 API、路径和方法的授权配置。

如果未指定 `DefaultAuthorizer`，则使用此属性覆盖单个路径的 API `DefaultAuthorizer` 设置，或者覆盖默认 `ApiKeyRequired` 设置。

类型：ApiStateMachineAuth

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`Method`

调用此函数的 HTTP 方法。

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`Path`

调用此函数的 URI 路径。值必须以 / 开头。

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`RestApiId`

`RestApi` 资源的标识符，必须包含具有给定路径和方法的操作。通常，将其设置为引用此模板中定义的 AWS::Serverless::Api 资源。

如果未定义此属性，则使用生成的`OpenApi`文档 Amazon SAM 创建默认AWS::Serverless::Api资源。该资源包含所有路径和方法的并集，这些路径和方法由同一模板中的 `Api` 事件定义，但未指定 `RestApiId`。

此属性无法引用在其他模板中定义的 AWS::Serverless::Api 资源。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`UnescapeMappingTemplate`

在传递给状态机的输入上，通过将 `\'` 替换为 `'` 来取消转义单引号。当输入包含单引号时使用。

> **Note**
> 如果设置为 `False` 并且输入包含单引号，则会发生错误。

类型：布尔值

必需：否

默认值：False

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

示例

ApiEvent

以下是 `Api` 类型事件的示例。

YAML

```
Events:
  ApiEvent:
    Type: Api
    Properties:
      Path: /path
      Method: get
```

ApiStateMachineAuth

在事件级别为特定 API、路径和方法配置授权。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
  ApiKeyRequired: Boolean
  AuthorizationScopes: List
  Authorizer: String
  ResourcePolicy: ResourcePolicyStatement
```

属性

`ApiKeyRequired`

此 API、路径和方法需要 API 密钥。

类型：布尔值

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`AuthorizationScopes`

适用于此 API、路径和方法的授权范围。

如果您已指定 `DefaultAuthorizer` 属性，则您指定的范围将覆盖该属性应用的所有范围。

类型：列表

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`Authorizer`

适用于特定状态机的 `Authorizer`。

如果您已为 API 指定了全局授权方并希望公开此状态机，请通过将 `Authorizer` 设置为 `NONE` 来覆盖全局授权方。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`ResourcePolicy`

为此 API 和路径配置资源策略。

类型：ResourcePolicyStatement

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

示例

StateMachine-身份验证

以下示例指定了状态机级别的授权。

YAML

```
Auth:
  ApiKeyRequired: true
```

```
  Authorizer: NONE
```

ResourcePolicyStatement

为 API 的所有方法和路径配置资源策略。有关资源策略的更多信息，请参阅《API Gateway 开发人员指南》中的使用 API Gateway 资源策略控制 API 的访问权限。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```yaml
  AwsAccountBlacklist: List
  AwsAccountWhitelist: List
  CustomStatements: List
  IntrinsicVpcBlacklist: List
  IntrinsicVpcWhitelist: List
  IntrinsicVpceBlacklist: List
  IntrinsicVpceWhitelist: List
  IpRangeBlacklist: List
  IpRangeWhitelist: List
  SourceVpcBlacklist: List
  SourceVpcWhitelist: List
```

属性

`AwsAccountBlacklist`

要封锁的 Amazon 账户。

类型：字符串列表

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`AwsAccountWhitelist`

要允许的 Amazon 账户。有关此属性的使用示例，请参阅本页底部的“示例”部分。

类型：字符串列表

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`CustomStatements`

适用于此 API 的自定义资源策略语句列表。有关此属性的使用示例，请参阅本页底部的“示例”部分。

类型：列表

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`IntrinsicVpcBlacklist`

要屏蔽的虚拟私有云列表 (VPCs)，其中每个 VPC 都指定为引用，例如动态引用或`Ref`内部函数。有关此属性的使用示例，请参阅本页底部的“示例”部分。

类型：列表

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`IntrinsicVpcWhitelist`

VPCs 要允许的列表，其中每个 VPC 都指定为引用，例如动态引用或`Ref`内部函数。

类型：列表

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`IntrinsicVpceBlacklist`

要阻止的 VPC 端点列表，其中每个 VPC 端点都指定为引用，例如动态引用或`Ref`内置函数。

类型：列表

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`IntrinsicVpceWhitelist`

要允许的 VPC 端点列表，其中每个 VPC 端点都指定为引用，例如动态引用或`Ref`内置函数。有关此属性的使用示例，请参阅本页底部的“示例”部分。

类型：列表

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`IpRangeBlacklist`

要阻止的 IP 地址或地址范围。有关此属性的使用示例，请参阅本页底部的“示例”部分。

类型：列表

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`IpRangeWhitelist`

要允许的 IP 地址或地址范围。

类型：列表

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`SourceVpcBlacklist`

要阻止的源 VPC 或 VPC 端点。源 VPC 名称必须以 `"vpc-"` 开头，源 VPC 端点名称必须以 `"vpce-"` 开头。有关此属性的使用示例，请参阅本页底部的“示例”部分。

类型：列表

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`SourceVpcWhitelist`

要允许的源 VPC 或 VPC 端点。源 VPC 名称必须以 `"vpc-"` 开头，源 VPC 端点名称必须以 `"vpce-"` 开头。

类型：列表

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

示例

资源策略示例

以下示例屏蔽两个 IP 地址和一个源 VPC，并允许一个 Amazon 账户。

YAML

```
Auth:
  ResourcePolicy:
    CustomStatements: [{
                        "Effect": "Allow",
                        "Principal": "*",
                        "Action": "execute-api:Invoke",
                        "Resource": "execute-api:/Prod/GET/pets",
                        "Condition": {
                          "IpAddress": {
                            "aws:SourceIp": "1.2.3.4"
                          }
                        }
                      }]
    IpRangeBlacklist:
      - "10.20.30.40"
      - "1.2.3.4"
    SourceVpcBlacklist:
      - "vpce-1a2b3c4d"
```

```
    AwsAccountWhitelist:
      - "111122223333"
    IntrinsicVpcBlacklist:
      - "{{resolve:ssm:SomeVPCReference:1}}"
      - !Ref MyVPC
    IntrinsicVpceWhitelist:
      - "{{resolve:ssm:SomeVPCEReference:1}}"
      - !Ref MyVPCE
```

CloudWatchEvent

描述 `CloudWatchEvent` 事件源类型的对象。

Amazon Serverless Application Model (Amazon SAM) 生成一个 AWS::Events::Rule设置此事件类型时的资源。

重要说明：EventBridgeRule是首选使用的事件源类型，而不是`CloudWatchEvent`。`EventBridgeRule`并`CloudWatchEvent`使用相同的底层服务、API 和 Amazon CloudFormation 资源。但是， Amazon SAM 将仅向添加对新功能的支持`EventBridgeRule`。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
EventBusName: String
Input: String
InputPath: String
Pattern: EventPattern
```

属性

`EventBusName`

要与该规则关联的事件总线。如果省略此属性，则 Amazon SAM 使用默认的事件总线。

类型：字符串

必需：否

默认：默认事件总线

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Events::Rule`资源的`EventBusName`属性。

`Input`

传递到目标的有效 JSON 文本。如果使用此属性，则不会将事件文本本身的任何内容传递到目标。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Events::Rule Target`资源的`Input`属性。

`InputPath`

当您不希望将整个匹配事件传递给目标时，请使用 `InputPath` 属性描述要传递事件的哪一部分。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Events::Rule Target`资源的`InputPath`属性。

`Pattern`

描述哪些事件路由到指定目标。有关更多信息，请参阅 Amazon EventBridge 用户指南 EventBridge中的事件和事件模式。

类型：EventPattern

必需：是

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Events::Rule`资源的`EventPattern`属性。

示例

CloudWatchEvent

以下是 `CloudWatchEvent` 事件源类型的示例。

YAML

```
CWEvent:
  Type: CloudWatchEvent
  Properties:
    Input: '{"Key": "Value"}'
    Pattern:
      detail:
        state:
          - running
```

EventBridgeRule

描述`EventBridgeRule`事件源类型的对象，它将您的状态机设置为 Amazon EventBridge 规则的目标。有关更多信息，请参阅什么是亚马逊 EventBridge？ 在《亚马逊 EventBridge 用户指南》中。

Amazon SAM 生成一个 AWS::Events::Rule设置此事件类型时的资源。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
  DeadLetterConfig: DeadLetterConfig
  EventBusName: String
  Input: String
  InputPath: String
  InputTransformer: InputTransformer
  Pattern: EventPattern
  RetryPolicy: RetryPolicy
  RuleName: String
  State: String
  Target: Target
```

属性

`DeadLetterConfig`

配置亚马逊简单队列服务 (Amazon SQS) Simple Queue Service 队列，目标调用失败后 EventBridge 在该队列中发送事件。例如，当向不存在的 Lambda 函数发送事件时，或者没有

足够的权限调用 Lambda 函数 EventBridge 时，调用可能会失败。有关更多信息，请参阅 A ma EventBridge zon 用户指南中的事件重试策略和使用死信队列。

类型：DeadLetterConfig

必需：否

Amazon CloudFormation 兼容性：此属性类似于`AWS::Events::RuleTarget`数据类型的`DeadLetterConfig`属性。此属性的 Amazon SAM 版本包括其他子属性， Amazon SAM 以备您想要创建死信队列时使用。

`EventBusName`

要与该规则关联的事件总线。如果省略此属性，则 Amazon SAM 使用默认的事件总线。

类型：字符串

必需：否

默认：默认事件总线

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Events::Rule`资源的`EventBusName`属性。

`Input`

传递到目标的有效 JSON 文本。如果使用此属性，则不会将事件文本本身的任何内容传递到目标。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Events::Rule Target`资源的`Input`属性。

`InputPath`

当您不希望将整个匹配事件传递给目标时，请使用 `InputPath` 属性描述要传递事件的哪一部分。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Events::Rule Target`资源的`InputPath`属性。

`InputTransformer`

使您可以根据特定事件数据向目标提供自定义输入的设置。您可以从事件中提取一个或多个键值对，然后使用该数据将自定义输入发送到目标。有关更多信息，请参阅《亚马逊 EventBridge 用户指南》中的亚马逊 EventBridge 输入转换。

类型：InputTransformer

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Events::RuleTarget`数据类型的`InputTransformer` 属性。

`Pattern`

描述哪些事件路由到指定目标。有关更多信息，请参阅 Amazon EventBridge 用户指南 EventBridge中的事件和事件模式。

类型：EventPattern

必需：是

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Events::Rule`资源的`EventPattern`属性。

`RetryPolicy`

包含有关重试策略设置的信息的 `RetryPolicy` 对象。有关更多信息，请参阅 A ma EventBridge zon 用户指南中的事件重试策略和使用死信队列。

类型：RetryPolicy

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Events::RuleTarget`数据类型的`RetryPolicy`属性。

`RuleName`

规则的名称。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Events::Rule`资源的`Name`属性。

`State`

规则的状态。

有效值：`[ DISABLED | ENABLED ]`

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Events::Rule`资源的`State`属性。

`Target`

触发规则时 EventBridge 调用的 Amazon 资源。您可以使用此属性来指定目标的逻辑 ID。如果未指定此属性，则 Amazon SAM 生成目标的逻辑 ID。

类型：目标

必需：否

Amazon CloudFormation 兼容性：此属性类似于`AWS::Events::Rule`资源的`Targets`属性。此属性的 Amazon SAM 版本仅允许您指定单个目标的逻辑 ID。

示例

EventBridgeRule

以下是 `EventBridgeRule` 事件源类型的示例。

YAML

```
EBRule:
  Type: EventBridgeRule
  Properties:
    Input: '{"Key": "Value"}'
    Pattern:
      detail:
```

```
        state:
          - terminated
```

DeadLetterConfig

用于指定亚马逊简单队列服务 (Amazon SQS) Simple Queue Service 队列的对象，目标调用失败后 EventBridge 在该队列中发送事件。例如，当向不存在的状态机发送事件或调用状态机的权限不足时，调用可能会失败。有关更多信息，请参阅 A ma EventBridge zon 用户指南中的事件重试策略和使用死信队列。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
Arn: String
QueueLogicalId: String
Type: String
```

属性

`Arn`

指定作为死信队列目标的 Amazon SQS 队列的 Amazon 资源名称（ARN）。

> **Note**
> 指定 `Type` 属性或 `Arn` 属性，但不能同时指定两者。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Events::RuleDeadLetterConfig`数据类型的`Arn`属性。

`QueueLogicalId`

指定了 Amazon SAM 创建的死信队列的自定义名称。`Type`

> **Note**
> 如果未设置 `Type` 属性，则将忽略该属性。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`Type`

队列的类型。设置此属性后， Amazon SAM 会自动创建死信队列并附加必要的基于资源的策略，以授予规则资源向队列发送事件的权限。

> **Note**
> 指定 `Type` 属性或 `Arn` 属性，但不能同时指定两者。

有效值：`SQS`

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

示例

DeadLetterConfig

DeadLetterConfig

YAML

```
DeadLetterConfig:
```

```
  Type: SQS
  QueueLogicalId: MyDLQ
```

Target

配置触发规则时 EventBridge 调用的 Amazon 资源。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
  Id: String
```

属性

`Id`

目标的逻辑 ID。

`Id` 的值可以包含字母数字字符、句点 (`.`)、连字符 (`-`) 和下划线 (`_`)。

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Events::RuleTarget`数据类型的`Id`属性。

示例

目标

YAML

```
EBRule:
  Type: EventBridgeRule
  Properties:
    Target:
      Id: MyTarget
```

Schedule

描述`Schedule`事件源类型的对象，它将您的状态机设置为按计划触发的 EventBridge 规则的目标。有关更多信息，请参阅什么是亚马逊 EventBridge？ 在《亚马逊 EventBridge 用户指南》中。

Amazon Serverless Application Model (Amazon SAM) 生成一个 AWS::Events::Rule设置此事件类型时的资源。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
DeadLetterConfig: DeadLetterConfig
Description: String
Enabled: Boolean
Input: String
Name: String
RetryPolicy: RetryPolicy
RoleArn: String
Schedule: String
State: String
Target: Target
```

属性

`DeadLetterConfig`

配置亚马逊简单队列服务 (Amazon SQS) Simple Queue Service 队列，目标调用失败后 EventBridge 在该队列中发送事件。例如，当向不存在的 Lambda 函数发送事件时，或者没有足够的权限调用 Lambda 函数 EventBridge 时，调用可能会失败。有关更多信息，请参阅 A ma EventBridge zon 用户指南中的事件重试策略和使用死信队列。

类型：DeadLetterConfig

必需：否

Amazon CloudFormation 兼容性：此属性类似于`AWS::Events::RuleTarget`数据类型的`DeadLetterConfig`属性。此属性的 Amazon SAM 版本包括其他子属性， Amazon SAM 以备您想要创建死信队列时使用。

`Description`

规则的描述。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Events::Rule`资源的`Description`属性。

`Enabled`

指示是否启用规则。

要禁用该规则，请将此属性设置为 `false`。

> **Note**
> 指定 `Enabled` 或 `State` 属性，但不能同时指定两者。

类型：布尔值

必需：否

Amazon CloudFormation 兼容性：此属性类似于`AWS::Events::Rule`资源的`State`属性。如果此属性设置为，`true`则 Amazon SAM 通过`ENABLED`，否则通过`DISABLED`。

`Input`

传递到目标的有效 JSON 文本。如果使用此属性，则不会将事件文本本身的任何内容传递到目标。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Events::Rule Target`资源的`Input`属性。

`Name`

规则的名称。如果您不指定名称，则 Amazon CloudFormation 会生成一个唯一的物理 ID 并将该 ID 用作规则名称。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Events::Rule`资源的`Name`属性。

`RetryPolicy`

包含有关重试策略设置的信息的 `RetryPolicy` 对象。有关更多信息，请参阅 A ma EventBridge zon 用户指南中的事件重试策略和使用死信队列。

类型：RetryPolicy

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Events::RuleTarget`数据类型的`RetryPolicy`属性。

`RoleArn`

调用计划时， EventBridge 计划程序将用于目标的 IAM 角色的 ARN。

类型：RoleArn

必需：否。如果未提供，则将创建并使用新角色。

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Scheduler::ScheduleTarget`数据类型的`RoleArn`属性。

`Schedule`

决定运行规则的时间和频率的计划表达式。有关更多信息，请参阅规则的计划表达式。

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Events::Rule`资源的`ScheduleExpression`属性。

`State`

规则的状态。

接受的值：`DISABLED | ENABLED`

> **Note**
>
> 指定 `Enabled` 或 `State` 属性，但不能同时指定两者。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Events::Rule`资源的`State`属性。

`Target`

触发规则时 EventBridge 调用的 Amazon 资源。您可以使用此属性来指定目标的逻辑 ID。如果未指定此属性，则 Amazon SAM 生成目标的逻辑 ID。

类型：目标

必需：否

Amazon CloudFormation 兼容性：此属性类似于`AWS::Events::Rule`资源的`Targets`属性。此属性的 Amazon SAM 版本仅允许您指定单个目标的逻辑 ID。

示例

CloudWatch 安排活动

CloudWatch 安排活动示例

YAML

```
CWSchedule:
  Type: Schedule
  Properties:
    Schedule: 'rate(1 minute)'
    Name: TestSchedule
    Description: test schedule
    Enabled: false
```

DeadLetterConfig

用于指定亚马逊简单队列服务 (Amazon SQS) Simple Queue Service 队列的对象，目标调用失败后 EventBridge 在该队列中发送事件。例如，当向不存在的状态机发送事件或调用状态机的权限不足时，

调用可能会失败。有关更多信息，请参阅 A ma EventBridge zon 用户指南中的事件重试策略和使用死信队列。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
Arn: String
QueueLogicalId: String
Type: String
```

属性

`Arn`

指定作为死信队列目标的 Amazon SQS 队列的 Amazon 资源名称（ARN）。

> **Note**
> 指定 `Type` 属性或 `Arn` 属性，但不能同时指定两者。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Events::RuleDeadLetterConfig`数据类型的`Arn`属性。

`QueueLogicalId`

指定了 Amazon SAM 创建的死信队列的自定义名称。`Type`

> **Note**
> 如果未设置 `Type` 属性，则将忽略该属性。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

Type

队列的类型。设置此属性后， Amazon SAM 会自动创建死信队列并附加必要的基于资源的策略，以授予规则资源向队列发送事件的权限。

> **Note**
>
> 指定 `Type` 属性或 `Arn` 属性，但不能同时指定两者。

有效值：`SQS`

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

示例

DeadLetterConfig

DeadLetterConfig

YAML

```
DeadLetterConfig:
  Type: SQS
  QueueLogicalId: MyDLQ
```

Target

配置触发规则时 EventBridge 调用的 Amazon 资源。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```
  Id: String
```

属性

`Id`

目标的逻辑 ID。

`Id` 的值可以包含字母数字字符、句点 (`.`)、连字符 (`-`) 和下划线 (`_`)。

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Events::RuleTarget`数据类型的`Id`属性。

示例

目标

YAML

```
EBRule:
  Type: Schedule
  Properties:
    Target:
      Id: MyTarget
```

ScheduleV2

描述`ScheduleV2`事件源类型的对象，它将您的状态机设置为按计划触发的 Amazon S EventBridge cheduler 事件的目标。有关更多信息，请参阅什么是 Amazon EventBridge 日程安排？ 在《EventBridge 日程安排器用户指南》中。

Amazon Serverless Application Model (Amazon SAM) 生成一个 AWS::Scheduler::Schedule设置此事件类型时的资源。

语法

要在 Amazon Serverless Application Model (Amazon SAM) 模板中声明此实体，请使用以下语法。

YAML

```yaml
DeadLetterConfig: DeadLetterConfig
Description: String
EndDate: String
FlexibleTimeWindow: FlexibleTimeWindow
GroupName: String
Input: String
KmsKeyArn: String
Name: String
OmitName: Boolean
PermissionsBoundary: String
RetryPolicy: RetryPolicy
RoleArn: String
ScheduleExpression: String
ScheduleExpressionTimezone: String
StartDate: String
State: String
```

属性

DeadLetterConfig

配置亚马逊简单队列服务 (Amazon SQS) Simple Queue Service 队列，目标调用失败后 EventBridge 在该队列中发送事件。例如，当向不存在的 Lambda 函数发送事件时，或者没有足够的权限调用 Lambda 函数 EventBridge 时，调用可能会失败。有关更多信息，请参阅《日 EventBridge 程安排器用户指南》中的为调度程序配置死信队列。EventBridge

类型：DeadLetterConfig

必需：否

Amazon CloudFormation 兼容性：此属性类似于`AWS::Scheduler::ScheduleTarget`数据类型的`DeadLetterConfig`属性。此属性的 Amazon SAM 版本包括其他子属性， Amazon SAM 以备您想要创建死信队列时使用。

Description

计划的描述。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Scheduler::Schedule`资源的`Description`属性。

`EndDate`

以 UTC 为单位的日期，在此日期之前计划可以调用其目标。根据计划的重复表达式，调用可能会在您指定的 EndDate 或之前停止。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Scheduler::Schedule`资源的`EndDate`属性。

`FlexibleTimeWindow`

允许配置可以调用计划的窗口。

类型：FlexibleTimeWindow

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Scheduler::Schedule`资源的`FlexibleTimeWindow`属性。

`GroupName`

将与此计划关联的计划组名称。如果未定义，则使用默认组。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Scheduler::Schedule`资源的`GroupName`属性。

`Input`

传递到目标的有效 JSON 文本。如果使用此属性，则不会将事件文本本身的任何内容传递到目标。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Scheduler::Schedule Target`资源的`Input`属性。

`KmsKeyArn`

将用于加密客户数据的 KMS 密钥的 ARN。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Scheduler::Schedule`资源的`KmsKeyArn`属性。

`Name`

计划的名称。如果您未指定名称，则 Amazon SAM 会生成格式为的名称，*`StateMachine-Logical-IDEvent-Source-Name`*并使用该 ID 作为计划名称。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Scheduler::Schedule`资源的`Name`属性。

`OmitName`

默认情况下， Amazon SAM 生成并使用格式为的计划名称*`<State-machine-logical-ID><event-source-name>`*。将此属性设置`true`为 Amazon CloudFormation 生成唯一的物理 ID，然后改用该物理 ID 作为计划名称。

类型：布尔值

必需：否

默认值：`false`

Amazon CloudFormation 兼容性：此属性是独有的 Amazon SAM ，没有 Amazon CloudFormation 等效属性。

`PermissionsBoundary`

用于为角色设置权限边界的策略的 ARN。

> **Note**
>
> 如果已定义，`PermissionsBoundary`则 Amazon SAM 将对计划程序计划的目标 IAM 角色应用相同的边界。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::IAM::Role`资源的`PermissionsBoundary`属性。

`RetryPolicy`

包含有关重试策略设置的信息的 `RetryPolicy` 对象。

类型：RetryPolicy

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Scheduler::ScheduleTarget`数据类型的`RetryPolicy`属性。

`RoleArn`

调用计划时， EventBridge 计划程序将用于目标的 IAM 角色的 ARN。

类型：RoleArn

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Scheduler::ScheduleTarget`数据类型的`RoleArn`属性。

`ScheduleExpression`

决定运行计划的时间和频率的计划表达式。

类型：字符串

必需：是

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Scheduler::Schedule`资源的`ScheduleExpression`属性。

`ScheduleExpressionTimezone`

评估计划表达式的时区。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Scheduler::Schedule`资源的`ScheduleExpressionTimezone`属性。

`StartDate`

以 UTC 为单位的日期，在此日期之后计划可以调用目标。根据计划的重复表达式，调用可能会在您指定的 StartDate 或之后发生。

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Scheduler::Schedule`资源的`StartDate`属性。

`State`

计划的状态。

接受的值：`DISABLED | ENABLED`

类型：字符串

必需：否

Amazon CloudFormation 兼容性：此属性直接传递给`AWS::Scheduler::Schedule`资源的`State`属性。

示例

定义 ScheduleV2 资源的基本示例

```
StateMachine:
  Type: AWS::Serverless::StateMachine
  Properties:
    Name: MyStateMachine
```

```
    Events:
      ScheduleEvent:
        Type: ScheduleV2
        Properties:
          ScheduleExpression: "rate(1 minute)"
      ComplexScheduleEvent:
        Type: ScheduleV2
        Properties:
          ScheduleExpression: rate(1 minute)
          FlexibleTimeWindow:
            Mode: FLEXIBLE
            MaximumWindowInMinutes: 5
          StartDate: '2022-12-28T12:00:00.000Z'
          EndDate: '2023-01-28T12:00:00.000Z'
          ScheduleExpressionTimezone: UTC
          RetryPolicy:
            MaximumRetryAttempts: 5
            MaximumEventAgeInSeconds: 300
          DeadLetterConfig:
            Type: SQS
    DefinitionUri:
      Bucket: sam-sam-s3-demo-bucket
      Key: my-state-machine.asl.json
      Version: 3
    Policies:
      - LambdaInvokePolicy:
          FunctionName: !Ref MyFunction
```

生成的 Amazon CloudFormation 资源用于 Amazon SAM

本节详细介绍了在 Amazon SAM 处理 Amazon 模板时创建的 Amazon CloudFormation 资源。根据您指定的方案， Amazon SAM 生成的 Amazon CloudFormation 资源集会有所不同。场景是在模板文件中指定的 Amazon SAM 资源和属性的组合。您可以在模板文件中的其他位置引用生成的 Amazon CloudFormation 资源，类似于引用在模板文件中明确声明的资源。

例如，如果您在 Amazon SAM 模板文件中指定了 `AWS::Serverless::Function` 资源，则 Amazon SAM 总是会生成 `AWS::Lambda::Function` 基本资源。如果您还指定了可选`AutoPublishAlias`属性，则 Amazon SAM 还会生成`AWS::Lambda::Alias`和`AWS::Lambda::Version`资源。

本节列出了场景及其生成的 Amazon CloudFormation 资源，并说明如何在 Amazon SAM 模板文件中引用生成的 Amazon CloudFormation 资源。

引用生成的 Amazon CloudFormation 资源

您可以通过`LogicalId`或按可引用属性在 Amazon SAM 模板文件中引用生成的 Amazon CloudFormation 资源。

通过以下方式引用生成的 Amazon CloudFormation 资源 LogicalId

Amazon SAM 生成每个 Amazon CloudFormation 资源的资源都有一个字母数字（A-Z `LogicalId`、a-z、0-9）标识符，在模板文件中是唯一的。 Amazon SAM 使用模板文件`LogicalIds`中的 Amazon SAM 资源来构造它生成的 Amazon CloudFormation 资源。`LogicalIds`您可以使用生成的 Amazon CloudFormation 资源在`LogicalId`模板文件中访问该资源的属性，就像访问已明确声明的 Amazon CloudFormation 资源一样。有关 Amazon CloudFormation 和 Amazon SAM 模板`LogicalIds`的更多信息，请参阅《Amazon CloudFormation 用户指南》中的资源。

> **Note**
>
> 某些生成的资源的 `LogicalIds` 包含唯一哈希值，以避免命名空间冲突。这些资源的 `LogicalIds` 是在创建堆栈时派生的。只有在创建堆栈之后，您才能使用 Amazon Web Services Management Console Amazon CLI、或其中一个来检索它们 Amazon SDKs。我们不建议通过 `LogicalId` 引用这些资源，因为哈希值可能会发生变化。

通过可引用属性引用生成的 Amazon CloudFormation 资源

对于某些生成的资源， Amazon SAM 提供该资源的可引用属性。 Amazon SAM 您可以使用此属性在 Amazon SAM 模板文件中引用生成的 Amazon CloudFormation 资源及其属性。

> **Note**
>
> 并非所有生成的 Amazon CloudFormation 资源都具有可引用的属性。对于这些资源，必须使用 `LogicalId`。

生成的 Amazon CloudFormation 资源场景

下表汇总了构成生成 Amazon SAM 资源的场景的 Amazon CloudFormation 资源和属性。场景列中的主题提供了有关为该场景 Amazon SAM 生成的其他 Amazon CloudFormation 资源的详细信息。

Amazon SAM 资源	基础 Amazon CloudFormation 资源	场景
AWS::Serverless::Api	AWS::ApiGateway::RestApi	• DomainName属性已指定 • UsagePlan属性已指定
AWS::Serverless::Application	AWS::CloudFormation::Stack	• 除了生成基础 Amazon CloudFormation 资源外，此无服务器资源没有其他方案。
AWS::Serverless::Function	AWS::Lambda::Function	• AutoPublishAlias 属性已指定 • 未指定 Role 属性 • DeploymentPreference 属性已指定 • 指定了 Api 事件源 • 已指定 HttpApi事件源 • 指定了流式事件源 • 指定了事件桥（或事件总线）事件源 • 已指定 IotRule事件源 • OnSuccess（或 OnFailure）属性是为亚马逊 SNS 事件指定的 • OnSuccess（或 OnFailure）属性是为亚马逊 SQS 事件指定的

Amazon SAM 资源	基础 Amazon CloudFormation 资源	场景
AWS::Serverless::HttpApi	AWS::ApiGatewayV2::Api	• StageName属性已指定 • StageName未指定属性 • DomainName属性已指定
AWS::Serverless::LayerVersion	AWS::Lambda::LayerVersion	• 除了生成基础 Amazon CloudFormation 资源外，此无服务器资源没有其他方案。
AWS::Serverless::SimpleTable	AWS::DynamoDB::Table	• 除了生成基础 Amazon CloudFormation 资源外，此无服务器资源没有其他方案。
AWS::Serverless::StateMachine	AWS::StepFunctions::StateMachine	• 未指定 Role 属性 • 指定了 API 事件源 • 指定了事件桥（或事件总线）事件源

主题

- Amazon CloudFormation 何时生成的资源 AWS::Serverless::Api 已指定
- Amazon CloudFormation 何时生成的资源 AWS::Serverless::Application 已指定
- Amazon CloudFormation 指定时生成的资源 AWS::Serverless::Connector
- Amazon CloudFormation 何时生成的资源 AWS::Serverless::Function 已指定
- Amazon CloudFormation 何时生成的资源 AWS::Serverless::GraphQLApi 已指定
- Amazon CloudFormation 指定时 AWS::Serverless::HttpApi 生成的资源
- Amazon CloudFormation 何时生成的资源 AWS::Serverless::LayerVersion 已指定

- Amazon CloudFormation 何时生成的资源 AWS::Serverless::SimpleTable 已指定
- Amazon CloudFormation 何时生成的资源 AWS::Serverless::StateMachine 已指定

Amazon CloudFormation 何时生成的资源 AWS::Serverless::Api 已指定

指定`AWS::Serverless::Api`时， Amazon Serverless Application Model (Amazon SAM) 始终生成`AWS::ApiGateway::RestApi`基础 Amazon CloudFormation 资源。此外，它还总是生成 `AWS::ApiGateway::Stage` 和 `AWS::ApiGateway::Deployment` 资源。

`AWS::ApiGateway::RestApi`

`LogicalId`: *`<api-LogicalId>`*

可引用的属性：N/A（必须使用`LogicalId`来引用此 Amazon CloudFormation 资源）

`AWS::ApiGateway::Stage`

`LogicalId`: *`<api-LogicalId><stage-name>`*`Stage`

`<stage-name>` 是 `StageName` 属性将被设置为的字符串。例如，您将 `StageName` 设置为 `Gamma`，`LogicalId` 是 *`MyRestApiGamma`*`Stage`。

可引用属性：*`<api-LogicalId>`*`.Stage`

`AWS::ApiGateway::Deployment`

`LogicalId`: *`<api-LogicalId>`*`Deployment`*`<sha>`*

`<sha>` 是在创建堆栈时生成的唯一哈希值。例如，*`MyRestApi`*`Deployment`*`926eeb5ff1`*。

可引用属性：*`<api-LogicalId>`*`.Deployment`

除了这些 Amazon CloudFormation 资源之外，如果`AWS::Serverless::Api`指定了这些资源，还会为以下场景 Amazon SAM 生成其他 Amazon CloudFormation 资源。

场景

- DomainName属性已指定
- UsagePlan属性已指定

DomainName属性已指定

指定DomainName属性的Domain属性后，将 Amazon SAM 生成AWS::ApiGateway::DomainName Amazon CloudFormation 资源。AWS::Serverless::Api

AWS::ApiGateway::DomainName

LogicalId: ApiGatewayDomainName*<sha>*

<sha> 是在创建堆栈时生成的唯一哈希值。例如：ApiGatewayDomainName*926eeb5ff1*。

可引用属性：*<api-LogicalId>*.DomainName

UsagePlan属性已指定

指定UsagePlan属性的Auth属性后， Amazon SAM 会生成以下 Amazon CloudFormation 资源：AWS::ApiGateway::UsagePlanAWS::ApiGateway::UsagePlanKey、和AWS::ApiGateway::ApiKey。AWS::Serverless::Api

AWS::ApiGateway::UsagePlan

LogicalId: *<api-LogicalId>*UsagePlan

可引用属性：*<api-LogicalId>*.UsagePlan

AWS::ApiGateway::UsagePlanKey

LogicalId: *<api-LogicalId>*UsagePlanKey

可引用属性：*<api-LogicalId>*.UsagePlanKey

AWS::ApiGateway::ApiKey

LogicalId: *<api-LogicalId>*ApiKey

可引用属性：*<api-LogicalId>*.ApiKey

Amazon CloudFormation 何时生成的资源 AWS::Serverless::Application 已指定

当指定AWS::Serverless::Application时， Amazon Serverless Application Model (Amazon SAM) 会生成AWS::CloudFormation::Stack基础 Amazon CloudFormation 资源。

AWS::CloudFormation::Stack

LogicalId: *<application-LogicalId>*

可引用的属性：N/A（必须使用`LogicalId`来引用此 Amazon CloudFormation 资源）

Amazon CloudFormation 指定时生成的资源 AWS::Serverless::Connector

> **Note**
>
> 如果通过嵌入式 `Connectors` 属性定义连接器，在生成这些资源之前，会先转换为 `AWS::Serverless::Connector` 资源。

在 Amazon SAM 模板中指定`AWS::Serverless::Connector`资源时， Amazon SAM 会根据需要生成以下 Amazon CloudFormation 资源。

AWS::IAM::ManagedPolicy

LogicalId:*<connector-LogicalId>*`Policy`

可引用属性：N/A（要引用此 Amazon CloudFormation 资源，必须使用。）`LogicalId`

AWS::SNS::TopicPolicy

LogicalId:*<connector-LogicalId>*`TopicPolicy`

可引用属性：N/A（要引用此 Amazon CloudFormation 资源，必须使用。）`LogicalId`

AWS::SQS::QueuePolicy

LogicalId:*<connector-LogicalId>*`QueuePolicy`

可引用属性：N/A（要引用此 Amazon CloudFormation 资源，必须使用。）`LogicalId`

AWS::Lambda::Permission

LogicalId:*<connector-LogicalId><permission>*`LambdaPermission`

<permission> 是 `Permissions` 属性指定的权限。例如，`Write`。

可引用属性：N/A（要引用此 Amazon CloudFormation 资源，必须使用。）`LogicalId`

Amazon CloudFormation 何时生成的资源 AWS::Serverless::Function 已指定

指定`AWS::Serverless::Function`时， Amazon Serverless Application Model (Amazon SAM) 始终会创建`AWS::Lambda::Function`基础 Amazon CloudFormation 资源。

`AWS::Lambda::Function`

`LogicalId`: *`<function-LogicalId>`*

可引用的属性：N/A（必须使用`LogicalId`来引用此 Amazon CloudFormation 资源）

除此 Amazon CloudFormation 资源外，如果指定了`AWS::Serverless::Function`此资源，Amazon SAM 还会生成用于以下场景的 Amazon CloudFormation 资源。

场景

- AutoPublishAlias 属性已指定
- 未指定 Role 属性
- DeploymentPreference 属性已指定
- 指定了 Api 事件源
- 已指定 HttpApi事件源
- 指定了流式事件源
- 指定了事件桥（或事件总线）事件源
- 已指定 IotRule事件源
- OnSuccess（或 OnFailure）属性是为亚马逊 SNS 事件指定的
- OnSuccess（或 OnFailure）属性是为亚马逊 SQS 事件指定的

AutoPublishAlias 属性已指定

当指定了的`AutoPublishAlias`属性`AWS::Serverless::Function`时， Amazon SAM 会生成以下 Amazon CloudFormation 资源：`AWS::Lambda::Alias`和`AWS::Lambda::Version`。

`AWS::Lambda::Alias`

`LogicalId`: *`<function-LogicalId>`*`Alias`*`<alias-name>`*

`<alias-name>` 是 `AutoPublishAlias` 将被设置为的字符串。例如，如果设置为 `AutoPublishAliaslive`，则`LogicalId`为：*`MyFunction`*别名*`live`*。

可引用属性：`<function-LogicalId>.Alias`

AWS::Lambda::Version

LogicalId: `<function-LogicalId>Version<sha>`

`<sha>` 是在创建堆栈时生成的唯一哈希值。例如，*MyFunction*版本*926eeb5ff1*。

可引用属性：`<function-LogicalId>.Version`

有关该AutoPublishAlias属性的更多信息，请参阅的 “属性” 部分 AWS::Serverless::Function。

未指定 Role 属性

如果未指定 a AWS::Serverless::Function 的Role属性，则 Amazon SAM 会生成AWS::IAM::Role Amazon CloudFormation 资源。

AWS::IAM::Role

LogicalId: `<function-LogicalId>Role`

可引用的属性：N/A（必须使用LogicalId来引用此 Amazon CloudFormation 资源）

DeploymentPreference 属性已指定

当指定了的DeploymentPreference属性AWS::Serverless::Function时， Amazon SAM 会生成以下资源 Amazon CloudFormation 资源：AWS::CodeDeploy::Application和AWS::CodeDeploy::DeploymentGroup。此外，如果未指定DeploymentPreference对象的Role属性， Amazon SAM 还会生成AWS::IAM::Role Amazon CloudFormation 资源。

AWS::CodeDeploy::Application

LogicalId: ServerlessDeploymentApplication

可引用的属性：N/A（必须使用LogicalId来引用此 Amazon CloudFormation 资源）

AWS::CodeDeploy::DeploymentGroup

LogicalId: `<function-LogicalId>DeploymentGroup`

可引用的属性：N/A（必须使用`LogicalId`来引用此 Amazon CloudFormation 资源）

`AWS::IAM::Role`

`LogicalId`: `CodeDeployServiceRole`

可引用的属性：N/A（必须使用`LogicalId`来引用此 Amazon CloudFormation 资源）

指定了 Api 事件源

如果的`Event`属性设置`AWS::Serverless::Function`为`Api`，但未指定该`RestApiId`属性，则 Amazon SAM 生成`AWS::ApiGateway::RestApi` Amazon CloudFormation 资源。

`AWS::ApiGateway::RestApi`

`LogicalId`: `ServerlessRestApi`

可引用的属性：N/A（必须使用`LogicalId`来引用此 Amazon CloudFormation 资源）

已指定 HttpApi事件源

如果的`Event`属性设置`AWS::Serverless::Function`为`HttpApi`，但未指定该`ApiId`属性，则 Amazon SAM 生成`AWS::ApiGatewayV2::Api` Amazon CloudFormation 资源。

`AWS::ApiGatewayV2::Api`

`LogicalId`: `ServerlessHttpApi`

可引用的属性：N/A（必须使用`LogicalId`来引用此 Amazon CloudFormation 资源）

指定了流式事件源

将的`Event`属性设置`AWS::Serverless::Function`为其中一种流媒体类型时， Amazon SAM 会生成`AWS::Lambda::EventSourceMapping` Amazon CloudFormation 资源。这适用于以下类型：`DynamoDB`、`Kinesis`、`MQ`、`MSK` 和 `SQS`。

`AWS::Lambda::EventSourceMapping`

`LogicalId`: *`<function-LogicalId><event-LogicalId>`*

可引用的属性：N/A（必须使用`LogicalId`来引用此 Amazon CloudFormation 资源）

指定了事件桥（或事件总线）事件源

将的`Event`属性设置`AWS::Serverless::Function`为事件桥（或事件总线）类型之一时，Amazon SAM 会生成`AWS::Events::Rule` Amazon CloudFormation 资源。这适用于以下类型：`EventBridgeRule`、`Schedule` 和 `CloudWatchEvents`。

`AWS::Events::Rule`

`LogicalId`: *`<function-LogicalId><event-LogicalId>`*

可引用的属性：N/A（必须使用`LogicalId`来引用此 Amazon CloudFormation 资源）

已指定 IotRule事件源

当的`Event`属性设置`AWS::Serverless::Function`为 Io 时TRule， Amazon SAM 会生成`AWS::IoT::TopicRule` Amazon CloudFormation 资源。

`AWS::IoT::TopicRule`

`LogicalId`: *`<function-LogicalId><event-LogicalId>`*

可引用的属性：N/A（必须使用`LogicalId`来引用此 Amazon CloudFormation 资源）

OnSuccess（或 OnFailure）属性是为亚马逊 SNS 事件指定的

如果指定了属性的属性`AWS::Serverless::Function`的`OnSuccess`（或`OnFailure`）`EventInvokeConfig`属性，且目标类型为，`SNS`但未指定目标 ARN，则 Amazon SAM 生成以下 Amazon CloudFormation 资源：`AWS::Lambda::EventInvokeConfig`和。`DestinationConfig` `AWS::SNS::Topic`

`AWS::Lambda::EventInvokeConfig`

`LogicalId`: *`<function-LogicalId>`*`EventInvokeConfig`

可引用的属性：N/A（必须使用`LogicalId`来引用此 Amazon CloudFormation 资源）

`AWS::SNS::Topic`

`LogicalId`：*`<function-LogicalId>`*`OnSuccessTopic`（或 *`<function-LogicalId>`*`OnFailureTopic`）

可引用属性：*`<function-LogicalId>`*`.DestinationTopic`

如果为 Amazon SNS 事件指定了 `OnSuccess` 和 `OnFailure`，为了区分生成的资源，必须使用 `LogicalId`。

OnSuccess（或 OnFailure）属性是为亚马逊 SQS 事件指定的

如果指定了属性的属性`AWS::Serverless::Function`的`OnSuccess`（或`OnFailure`）`EventInvokeConfig`属性，且目标类型为，`SQS`但未指定目标 ARN，则 Amazon SAM 生成以下 Amazon CloudFormation 资源：`AWS::Lambda::EventInvokeConfig`和。`DestinationConfig` `AWS::SQS::Queue`

`AWS::Lambda::EventInvokeConfig`

`LogicalId`: *`<function-LogicalId>`*`EventInvokeConfig`

可引用的属性：N/A（必须使用`LogicalId`来引用此 Amazon CloudFormation 资源）

`AWS::SQS::Queue`

`LogicalId`：*`<function-LogicalId>`*`OnSuccessQueue`（或 *`<function-LogicalId>`*`OnFailureQueue`）

可引用属性：*`<function-LogicalId>`*`.DestinationQueue`

如果为 Amazon SQS 事件指定了 `OnSuccess` 和 `OnFailure`，为了区分生成的资源，必须使用 `LogicalId`。

Amazon CloudFormation 何时生成的资源 AWS::Serverless::GraphQLApi 已指定

在 Amazon Serverless Application Model (Amazon SAM) 模板中指定`AWS::Serverless::GraphQLApi`资源时， Amazon SAM 始终会创建以下基础 Amazon CloudFormation 资源。

`AWS::AppSync::DataSource`

`LogicalId`: *`<graphqlapi-LogicalId><datasource-RelativeId><datasource-Type>`*`DataSource`

可引用的属性：N/A（必须使用`LogicalId`来引用此 Amazon CloudFormation 资源）

`AWS::AppSync::FunctionConfiguration`

`LogicalId`: *`<graphqlapi-LogicalId><function-RelativeId>`*

可引用的属性：N/A（必须使用`LogicalId`来引用此 Amazon CloudFormation 资源）

`AWS::AppSync::GraphQLApi`

`LogicalId`: *`<graphqlapi-LogicalId>`*

可引用的属性：N/A（必须使用`LogicalId`来引用此 Amazon CloudFormation 资源）

`AWS::AppSync::GraphQLSchema`

`LogicalId`: *`<graphqlapi-LogicalId>`*`Schema`

可引用的属性：N/A（必须使用`LogicalId`来引用此 Amazon CloudFormation 资源）

`AWS::AppSync::Resolver`

`LogicalId`: *`<graphqlapi-LogicalId><OperationType><resolver-RelativeId>`*

可引用的属性：N/A（必须使用`LogicalId`来引用此 Amazon CloudFormation 资源）

除了这些 Amazon CloudFormation 资源之外，`AWS::Serverless::GraphQLApi`如果指定了这些资源，还 Amazon SAM 可能生成以下 Amazon CloudFormation 资源。

`AWS::AppSync::ApiCache`

`LogicalId`: *`<graphqlapi-LogicalId>`*`ApiCache`

可引用的属性：N/A（必须使用`LogicalId`来引用此 Amazon CloudFormation 资源）

`AWS::AppSync::ApiKey`

`LogicalId`: *`<graphqlapi-LogicalId><apikey-RelativeId>`*

可引用的属性：N/A（必须使用`LogicalId`来引用此 Amazon CloudFormation 资源）

`AWS::AppSync::DomainName`

`LogicalId`: *`<graphqlapi-LogicalId>`*`DomainName`

可引用的属性：N/A（必须使用`LogicalId`来引用此 Amazon CloudFormation 资源）

`AWS::AppSync::DomainNameApiAssociation`

`LogicalId`: *`<graphqlapi-LogicalId>`*`DomainNameApiAssociation`

可引用的属性：N/A（必须使用`LogicalId`来引用此 Amazon CloudFormation 资源）

Amazon SAM 也可以使用该`AWS::Serverless::Connector`资源来配置权限。有关更多信息，请参阅 Amazon CloudFormation 指定时生成的资源 AWS::Serverless::Connector。

Amazon CloudFormation 指定时 AWS::Serverless::HttpApi 生成的资源

当指定`AWS::Serverless::HttpApi`时， Amazon Serverless Application Model (Amazon SAM) 会生成`AWS::ApiGatewayV2::Api`基础 Amazon CloudFormation 资源。

`AWS::ApiGatewayV2::Api`

`LogicalId`: *`<httpapi-LogicalId>`*

可引用的属性：N/A（必须使用`LogicalId`来引用此 Amazon CloudFormation 资源）

除此 Amazon CloudFormation 资源外，如果指定了`AWS::Serverless::HttpApi`此资源， Amazon SAM 还会生成用于以下场景的 Amazon CloudFormation 资源：

场景

- StageName属性已指定
- StageName未指定属性
- DomainName属性已指定

StageName属性已指定

当指定了的`StageName`属性`AWS::Serverless::HttpApi`时， Amazon SAM 会生成`AWS::ApiGatewayV2::Stage` Amazon CloudFormation 资源。

`AWS::ApiGatewayV2::Stage`

`LogicalId`: *`<httpapi-LogicalId><stage-name>`*`Stage`

`<stage-name>` 是 `StageName` 属性将被设置为的字符串。例如，如果设置为 `StageNameGamma`，则`LogicalId`为：*`MyHttpApiGamma`*舞台。

可引用属性：*`<httpapi-LogicalId>`*`.Stage`

StageName未指定属性

如果未指定 a `AWS::Serverless::HttpApi` 的`StageName`属性，则 Amazon SAM 生成`AWS::ApiGatewayV2::Stage` Amazon CloudFormation 资源。

AWS::ApiGatewayV2::Stage

LogicalId: *<httpapi-LogicalId>*ApiGatewayDefaultStage

可引用属性：*<httpapi-LogicalId>*.Stage

DomainName属性已指定

当指定了Domain属性的属性AWS::Serverless::HttpApi时， Amazon SAM 会生成AWS::ApiGatewayV2::DomainName Amazon CloudFormation 资源。DomainName

AWS::ApiGatewayV2::DomainName

LogicalId: ApiGatewayDomainNameV2*<sha>*

<sha> 是在创建堆栈时生成的唯一哈希值。例如，ApiGatewayDomainNameV2*926eeb5ff1*。

可引用属性：*<httpapi-LogicalId>*.DomainName

Amazon CloudFormation 何时生成的资源 AWS::Serverless::LayerVersion 已指定

当指定AWS::Serverless::LayerVersion时， Amazon Serverless Application Model (Amazon SAM) 会生成AWS::Lambda::LayerVersion基础 Amazon CloudFormation 资源。

AWS::Lambda::LayerVersion

LogicalId: *<layerversion-LogicalId>*

可引用的属性：N/A（必须使用LogicalId来引用此 Amazon CloudFormation 资源）

Amazon CloudFormation 何时生成的资源 AWS::Serverless::SimpleTable 已指定

当指定AWS::Serverless::SimpleTable时， Amazon Serverless Application Model (Amazon SAM) 会生成AWS::DynamoDB::Table基础 Amazon CloudFormation 资源。

AWS::DynamoDB::Table

LogicalId: *<simpletable-LogicalId>*

可引用的属性：N/A（必须使用LogicalId来引用此 Amazon CloudFormation 资源）

Amazon CloudFormation 何时生成的资源 AWS::Serverless::StateMachine 已指定

如果指定了 `AWS::Serverless::StateMachine`， Amazon Serverless Application Model (Amazon SAM) 会生成 `AWS::StepFunctions::StateMachine` 基本 Amazon CloudFormation 资源。

`AWS::StepFunctions::StateMachine`

`LogicalId`: *`<statemachine-LogicalId>`*

可引用的属性：N/A（必须使用`LogicalId`来引用此 Amazon CloudFormation 资源）

除此 Amazon CloudFormation 资源外，如果指定了`AWS::Serverless::StateMachine`此资源，Amazon SAM 还会生成用于以下场景的 Amazon CloudFormation 资源：

场景

- 未指定 Role 属性
- 指定了 API 事件源
- 指定了事件桥（或事件总线）事件源

未指定 Role 属性

如果未指定 a `AWS::Serverless::StateMachine` 的`Role`属性，则 Amazon SAM 会生成`AWS::IAM::Role` Amazon CloudFormation 资源。

`AWS::IAM::Role`

`LogicalId`: *`<statemachine-LogicalId>`*`Role`

可引用的属性：N/A（必须使用`LogicalId`来引用此 Amazon CloudFormation 资源）

指定了 API 事件源

如果的`Event`属性设置`AWS::Serverless::StateMachine`为`Api`，但未指定该`RestApiId`属性，则 Amazon SAM 生成`AWS::ApiGateway::RestApi` Amazon CloudFormation 资源。

`AWS::ApiGateway::RestApi`

`LogicalId`: `ServerlessRestApi`

可引用的属性：N/A（必须使用`LogicalId`来引用此 Amazon CloudFormation 资源）

指定了事件桥（或事件总线）事件源

将的`Event`属性设置`AWS::Serverless::StateMachine`为事件桥（或事件总线）类型之一时， Amazon SAM 会生成`AWS::Events::Rule` Amazon CloudFormation 资源。这适用于以下类型：`EventBridgeRule`、`Schedule` 和 `CloudWatchEvents`。

`AWS::Events::Rule`

`LogicalId`: *`<statemachine-LogicalId><event-LogicalId>`*

可引用的属性：N/A（必须使用`LogicalId`来引用此 Amazon CloudFormation 资源）

支持的资源属性 Amazon SAM

资源属性是您可以添加的属 Amazon SAM 性和用于控制其他行为和关系的 Amazon CloudFormation 资源。有关资源属性的更多信息，请参阅《Amazon CloudFormation 用户指南》中的资源属性引用。

Amazon SAM 支持由定义的资源属性的子集 Amazon CloudFormation。在支持的资源属性中，有些属性仅复制到相应 Amazon CloudFormation 资源的基础生成的 Amazon SAM 资源中，有些则复制到由相应 Amazon CloudFormation 资源生成的所有生成的 Amazon SAM 资源中。有关从相应 Amazon CloudFormation 资源生成的 Amazon SAM 资源的更多信息，请参阅生成的 Amazon CloudFormation 资源用于 Amazon SAM。

下表汇总了按 Amazon SAM以下所异常列内容提供的资源属性支持。

资源属性	目标生成的资源
DependsOn 元数据 [1, 2]	仅限基础 Amazon CloudFormation 生成的资源。有关 Amazon SAM 资源和基础 Amazon CloudFormation 资源之间映射的信息，请参阅生成的 Amazon CloudFormation 资源场景。
Condition DeletionPolicy UpdateReplacePolicy	所有从相应 Amazon CloudFormation 资源生成的 Amazon SAM 资源。有关生成 Amazon CloudFormation 资源的场景的信息，请参阅生成的 Amazon CloudFormation 资源场景。

备注：

1. 有关使用 `Metadata` 资源属性和 `AWS::Serverless::Function` 资源类型的更多信息，请参阅 在中使用自定义运行时构建 Lambda 函数 Amazon SAM。
2. 有关使用 `Metadata` 资源属性和 `AWS::Serverless::LayerVersion` 资源类型的更多信息，请参阅 在中构建 Lambda 图层 Amazon SAM。

异常

前面描述的资源属性规则有许多例外情况：

- 对于`AWS::Lambda::LayerVersion`， Amazon SAM仅限自定义字段`DeletionPolicy`为生成的 Amazon CloudFormation 资源`RetentionPolicy`设置。它的优先级高于 `DeletionPolicy` 其本身。如果两者均未设置，则默认将 `DeletionPolicy` 设置为 `Retain`。
- 对于 `AWS::Lambda::Version`，如果未指定 `DeletionPolicy`，则默认为 `Retain`。
- 对于为无服务器函数指定的场`DeploymentPreferences`景，资源属性不会复制到以下生成的 Amazon CloudFormation 资源中：
  - `AWS::CodeDeploy::Application`
  - `AWS::CodeDeploy::DeploymentGroup`
  - 为此场景创建的名为 `CodeDeployServiceRole` 的 `AWS::IAM::Role`
- 如果您的 Amazon SAM 模板包含多个带有隐式创建的 API 事件源的函数，则这些函数将共享生成的`AWS::ApiGateway::RestApi`资源。在这种情况下，如果函数具有不同的资源属性，则对于生成的`AWS::ApiGateway::RestApi`资源，根据以下优先级列表 Amazon SAM 复制资源属性：
  - `UpdateReplacePolicy`:
    1. `Retain`
    2. `Snapshot`
    3. `Delete`
  - `DeletionPolicy`:
    1. `Retain`
    2. `Delete`

适用于 API Gateway 的扩展 Amazon SAM

API Gateway 扩展专为 Amazon设计而设计，为设计和管理 APIs提供了额外的自定义设置和功能。API Gateway 扩展是 OpenAPI 规范的扩展，它支持 Amazon特定授权和特定于 API 网关的 API 集成。有关 API Gateway 扩展的更多信息，请参阅基于 OpenAPI 的 API Gateway 扩展。

Amazon SAM 支持 API Gateway 扩展的子集。要查看支持哪些 API Gateway 扩展 Amazon SAM，请参阅下表。

API Gateway 扩展	由... 支持 Amazon SAM
x-amazon-apigateway-any-method对象	是
x-amazon-apigateway-api-key-source 属性	否
x-amazon-apigateway-auth Object	是
x-amazon-apigateway-authorizer Object	是
x-amazon-apigateway-authtype 属性	是
x-amazon-apigateway-binary-媒体类型属性	是
x-amazon-apigateway-documentation Object	否
x-amazon-apigateway-endpoint-配置对象	否
x-amazon-apigateway-gateway-响应对象	是
x-amazon-apigateway-gateway-responses.gatewayRespon	是
x-amazon-apigateway-gateway-responses.responseparameter	是
x-amazon-apigateway-gateway-responses.responseTemp	是
x-amazon-apigateway-integration Object	是
x-amazon-apigateway-integration.requestTemp	是
x-amazon-apigateway-integration. 请求参数对象	否

x-amazon-apigateway-integration. 响应对象	是
x-amazon-apigateway-integration.response	是
x-amazon-apigateway-integration.responseTempl	是
x-amazon-apigateway-integration.responseparameters	是
x-amazon-apigateway-request-验证器属性	否
x-amazon-apigateway-request-验证器对象	否
x-amazon-apigateway-request-validators.requestValidator 对象	否

的内在函数 Amazon SAM

内部函数是内置函数，允许您为仅在运行时可用的属性赋值。 Amazon SAM 对某些内部函数属性的支持有限，因此它无法解析某些内部函数。因此，建议添加 `AWS::LanguageExtensions` 转换来解决此问题。是由托管的宏 Amazon CloudFormation ，允许您使用默认情况下不包含的内部函数和其他功能。`AWS::LanguageExtensions` Amazon CloudFormation

```
Transform:
  - AWS::LanguageExtensions
  - AWS::Serverless-2016-10-31
```

> **Note**
>
> 注意：如果你在 CodeUri 属性中使用内部函数， Amazon SAM 将无法正确解析这些值。考虑改用 `AWS::LanguageExtensions` 转换。
> 有关更多信息，请参阅的 “属性” 部分 AWS::Serverless::Function。

有关内置函数的更多信息，请参阅《Amazon CloudFormation 用户指南》中的内置函数参考。

使用开发您的无服务器应用程序 Amazon SAM

本节包含有关验证 Amazon SAM 模板和使用依赖项构建应用程序的主题。它还包含有关在某些用例中使用的主题，例如使用 Lambda 层、使用嵌套应用程序、控制 API Gat APIs eway 的访问权限、使用 Ste Amazon p Functions 编排资源以及对应用程序进行代码签名。 Amazon SAM 下面列出了开发应用程序需要完成的三个主要里程碑。

主题

- 在中创建您的应用程序 Amazon SAM
- 使用以下方式定义您的基础架构 Amazon SAM
- 使用构建您的应用程序 Amazon SAM

在中创建您的应用程序 Amazon SAM

完成入门和阅读如何使用 Amazon Serverless Application Model (Amazon SAM)后，您就可以准备好在开发人员环境中创建 Amazon SAM 项目了。您的 Amazon SAM 项目将作为编写无服务器应用程序的起点。有关清单 Amazon SAM CLI `sam init`命令选项，请参阅sam init。

Amazon Serverless Application Model 命令行界面 (Amazon SAM CLI) `sam init` 命令提供了用于初始化新的无服务器应用程序的选项，该应用程序包括：

- 用于定义基础架构代码的 Amazon SAM 模板。
- 用于组织应用程序的文件夹结构。
- 为您的 Amazon Lambda 函数进行配置。

要创建 Amazon SAM 项目，请参阅本节中的主题。

主题

- 初始化新的无服务器应用程序
- sam init 的选项
- 故障排除
- 示例
- 了解更多
- 后续步骤

初始化新的无服务器应用程序

要初始化新的无服务器应用程序，请使用 Amazon SAM CLI

1. 指向起始目录的 cd。
2. 在命令行中执行以下命令：

```
$ sam init
```

3. 的 Amazon SAM CLI 将引导您完成交互式流程，以创建新的无服务器应用程序。

> **Note**
>
> 如教程：使用以下命令部署 Hello World 应用程序 Amazon SAM中所述，此命令会初始化您的无服务器应用程序，从而创建您的项目目录。此目录将包含多个文件和文件夹。最重要的文件是 `template.yaml`。这是你的 Amazon SAM 模板。您的 python 版本必须与 sam init 命令创建的 `template.yaml` 文件中列出的 python 版本匹配。

选择起始模板

模板由以下部分组成：

1. 基础架构代码的 Amazon SAM 模板。
2. 用于组织项目文件的项目起始目录。例如，这可能包括：
   a. Lambda 函数代码及其依赖项的结构。
   b. 包含用于本地测试的测试事件的 `events` 文件夹。
   c. 支持单元测试的 `tests` 文件夹。
   d. 用于配置项目设置的 `samconfig.toml` 文件。
   e. `ReadMe` 文件和其他基本的项目起始文件。

   以下是项目起始目录的示例：

```
sam-app
### README.md
### __init__.py
### events
#   ### event.json
```

```
### hello_world
#   ### __init__.py
#   ### app.py
#   ### requirements.txt
### samconfig.toml
### template.yaml
### tests
    ### __init__.py
    ### integration
    #   ### __init__.py
    #   ### test_api_gateway.py
    ### requirements.txt
    ### unit
        ### __init__.py
        ### test_handler.py
```

您可以从可用的 Amazon 快速入门模板列表中进行选择，也可以自行提供自定义模板位置。

选择 Amazon 快速入门模板

1. 当系统提示时，选择 Amazon 快速入门模板。
2. 首先选择一个 Amazon 快速入门模板。以下是示例：

```
Which template source would you like to use?
    1 - AWS Quick Start Templates
    2 - Custom Template Location
Choice: 1

Choose an AWS Quick Start application template
    1 - Hello World Example
    2 - Multi-step workflow
    3 - Serverless API
    4 - Scheduled task
    5 - Standalone function
    6 - Data processing
    7 - Hello World Example With Powertools
    8 - Infrastructure event management
    9 - Serverless Connector Hello World Example
    10 - Multi-step workflow with Connectors
    11 - Lambda EFS example
    12 - DynamoDB Example
    13 - Machine Learning
```

```
Template: 4
```

选择自定义模板位置

1. 当系统提示时，选择自定义模板位置。

```
Which template source would you like to use?
    1 - AWS Quick Start Templates
    2 - Custom Template Location
Choice: 2
```

2. 的 Amazon SAM CLI 将提示您提供模板位置。

```
Template location (git, mercurial, http(s), zip, path):
```

为模板 .zip 文件存档提供以下任意位置：

- GitHub 存储库 -您的.zip 文件的路径 GitHub 存储库。文件必须位于存储库的根目录中。
- Mercurial 存储库 -您的.zip 文件的路径 Mercurial 存储库。文件必须位于存储库的根目录中。
- .zip 路径 - .zip 文件的 HTTPS 或本地路径。

3. 的 Amazon SAM CLI 将使用您的自定义模板初始化您的无服务器应用程序。

选择运行时系统

选择Amazon 快速入门模板时， Amazon SAM CLI 提示您为 Lambda 函数选择运行时。显示的选项列表 Amazon SAM CLI 是 Lambda 原生支持的运行时。

- 运行时提供在执行环境中运行的语言特定环境。
- 部署到后 Amazon Web Services 云，Lambda 服务将在执行环境中调用您的函数。

您可以将任何其他编程语言与自定义运行时配合使用。若要这样做，您需要手动创建应用程序起始结构。然后，您可以配置自定义模板位置，使用 `sam init` 来快速初始化应用程序。

根据您的选择， Amazon SAM CLI 为您的 Lambda 函数代码和依赖项创建起始目录。

如果 Lambda 对于运行时支持多个依赖项管理器，则系统会提示您选择首选的依赖项管理器。

选择包类型

选择Amazon 快速入门模板和运行时时， Amazon SAM CLI 会提示您选择包裹类型。包类型决定了如何部署 Lambda 函数以与 Lambda 服务配合使用。受支持的两种包类型是：

1. 容器印象 – 包括基本操作系统、运行时系统、Lambda 扩展、应用程序代码及其依赖项。
2. .zip 文件归档 – 包括您的应用程序代码及其依赖项。

要了解有关部署包类型的更多信息，请参阅《Amazon Lambda 开发人员指南》中的 Lambda 部署包。

以下示例显示了将 Lambda 函数打包为容器映像的应用程序的目录结构。的 Amazon SAM CLI 下载图像并在函数的目录`Dockerfile`中创建一个来指定图像。

```
sam-app
### README.md
### __init__.py
### events
#   ### event.json
### hello_world
#   ### Dockerfile
#   ### __init__.py
#   ### app.py
#   ### requirements.txt
### samconfig.toml
### template.yaml
### tests
    ### __init__.py
    ### unit
        ### __init__.py
        ### test_handler.py
```

以下示例显示了将函数打包为 .zip 文件存档的应用程序的目录结构。

```
sam-app
### README.md
### __init__.py
### events
#   ### event.json
### hello_world
#   ### __init__.py
```

```
#   ### app.py
#   ### requirements.txt
### samconfig.toml
### template.yaml
### tests
    ### __init__.py
    ### integration
    #   ### __init__.py
    #   ### test_api_gateway.py
    ### requirements.txt
    ### unit
        ### __init__.py
        ### test_handler.py
```

配置 Amazon X-Ray 跟踪

您可以选择激活跟 Amazon X-Ray 踪。要了解更多信息，请参阅什么是 Amazon X-Ray？ 在《Amazon X-Ray 开发人员指南》中。

如果你激活， Amazon SAM CLI 配置您的 Amazon SAM 模板。以下是示例：

```
Globals:
  Function:
    ...
    Tracing: Active
  Api:
    TracingEnabled: True
```

使用 Amazon CloudWatch 应用程序见解配置监控

您可以选择使用 Amazon App CloudWatch lication Insights 激活监控。要了解更多信息，请参阅《亚马逊 CloudWatch 用户指南》中的 “亚马逊 CloudWatch 应用程序见解”。

如果你激活， Amazon SAM CLI 配置您的 Amazon SAM 模板。以下是示例：

```
Resources:
  ApplicationResourceGroup:
    Type: AWS::ResourceGroups::Group
    Properties:
      Name:
        Fn::Join:
        - ''
```

```
        - - ApplicationInsights-SAM-
          - Ref: AWS::StackName
      ResourceQuery:
        Type: CLOUDFORMATION_STACK_1_0
  ApplicationInsightsMonitoring:
    Type: AWS::ApplicationInsights::Application
    Properties:
      ResourceGroupName:
        Fn::Join:
        - ''
        - - ApplicationInsights-SAM-
          - Ref: AWS::StackName
      AutoConfigurationEnabled: 'true'
    DependsOn: ApplicationResourceGroup
```

命名您的应用程序

提供应用程序的名称。的 Amazon SAM CLI 使用此名称为您的应用程序创建顶级文件夹。

sam init 的选项

以下是可与 `sam init` 命令配合使用的一些主要选项。有关全部选项的列表，请参阅 sam init。

使用自定义模板位置初始化应用程序

使用 `--location` 选项并提供受支持的自定义模板位置。以下是示例：

```
$ sam init --location https://github.com/aws-samples/sessions-with-aws-sam/raw/master/starter-templates/web-app.zip
```

在不使用交互式流程的情况下初始化应用程序

使用 `--no-interactive` 选项并在命令行中提供您的配置选择，以跳过交互式流程。以下是示例：

```
$ sam init --no-interactive --runtime go1.x --name go-demo --dependency-manager mod --app-template hello-world
```

故障排除

要排除故障 Amazon SAM CLI，请参阅 Amazon SAM CLI 故障排除。

示例

使用 Hello World Amazon 入门模板初始化新的无服务器应用程序

有关此示例的更多信息，请参阅教程：部署 Hello World 应用程序中的 第 1 步：初始化示例 Hello World 应用程序。

使用自定义模板位置初始化新的无服务器应用程序

以下是提供以下内容的示例 GitHub 您的自定义模板的位置：

```
$ sam init --location gh:aws-samples/cookiecutter-aws-sam-python
$ sam init --location git+sh://git@github.com/aws-samples/cookiecutter-aws-sam-
python.git
$ sam init --location hg+ssh://hg@bitbucket.org/repo/template-name
```

以下是本地文件路径的示例：

```
$ sam init --location /path/to/template.zip
```

以下是 HTTPS 可访问路径的示例：

```
$ sam init --location https://github.com/aws-samples/sessions-with-aws-sam/raw/master/
starter-templates/web-app.zip
```

了解更多

要了解有关使用 `sam init` 命令的更多信息，请参阅以下内容：

- 学习 Amazon SAM：sam init — Serverless Land “学习 Amazon SAM” 系列开启 YouTube.
- 构造用于的无服务器应用程序 Amazon SAM CLI（与 SAM S2E7 的会话）— 开启系列的会议 Amazon SAM YouTube.

后续步骤

既然您已经创建了 Amazon SAM 项目，就可以开始创作应用程序了。有关执行此操作所需完成的任务的详细说明，请参阅使用以下方式定义您的基础架构 Amazon SAM。

使用以下方式定义您的基础架构 Amazon SAM

既然您已经创建了项目，就可以用定义应用程序基础架构了 Amazon SAM。通过配置 Amazon SAM 模板来定义应用程序的资源和属性（即 Amazon SAM 项目中的`template.yaml`文件）来实现此目的。

本节中的主题提供了有关在 Amazon SAM 模板（您的`template.yaml`文件）中定义基础架构的内容。它还包含有关为特定用例定义资源的主题，例如使用 Lambda 层、使用嵌套应用程序、控制 API Gat APIs eway 的访问权限、使用 Ste Amazon p Functions 编排资源、对应用程序进行代码签名以及验证模板。 Amazon SAM

主题

- 在 Amazon SAM 模板中定义应用程序资源
- 在 Amazon SAM 模板中设置和管理资源访问权限
- 使用您的 Amazon SAM 模板控制 API 访问权限
- 使用 Lambda 图层提高效率 Amazon SAM
- 使用嵌套应用程序重用代码和资源 Amazon SAM
- 使用“ EventBridge调度器”管理基于时间的事件 Amazon SAM
- 使用编排资源 Amazon SAMAmazon Step Functions
- 为您的 Amazon SAM 应用程序设置代码签名
- 验证 Amazon SAM 模板文件

在 Amazon SAM 模板中定义应用程序资源

您可以在 Amazon SAM 模板的`Resources`部分中定义您的无服务器应用程序使用的 Amazon 资源。在定义资源时，您可以确定该资源是什么、它如何与其他资源交互以及如何访问该资源（即资源的权限）。

Amazon SAM 模板的`Resources`部分可以包含 Amazon CloudFormation 资源和 Amazon SAM 资源的组合。此外，您可以将 Amazon SAM的简写语法用于以下资源：

Amazon SAM 简写语法	它如何处理相关 Amazon 资源
AWS::Serverless::Api	创建一组可通过 HTTPS 端点调用的 API Gateway 资源和方法。

Amazon SAM 简写语法	它如何处理相关 Amazon 资源
AWS::Serverless::Application	将来自 Amazon Serverless Application Repository 或来自 Amazon S3 存储桶的无服务器应用程序嵌入为嵌套应用程序。
AWS::Serverless::Connector	配置两种资源之间的权限。有关连接器的简介，请参阅使用 Amazon SAM 连接器管理资源权限。
AWS::Serverless::Function	创建触发该 Amazon Lambda 函数的函数、Amazon Identity and Access Management (IAM) 执行角色和事件源映射。
AWS::Serverless::GraphQLApi	创建并配置 Amazon AppSync GraphQL 您的无服务器应用程序的 API。
AWS::Serverless::HttpApi	创建 Amazon API Gateway HTTP API，这使您能够以比 REST 更低的延迟和更低的成本 RESTful APIs 进行创作 APIs。
AWS::Serverless::LayerVersion	创建包含 Lambda 函数 LayerVersion 所需的库或运行时代码的 Lambda。
AWS::Serverless::SimpleTable	创建具有单个属性主键的 DynamoDB 表。
AWS::Serverless::StateMachine	创建 Amazon Step Functions 状态机，您可以使用它来编排 Amazon Lambda 函数和其他 Amazon 资源，以形成复杂而强大的工作流程。

上述资源也列在 Amazon SAM 资源和财产中。

有关所有 Amazon 资源和属性类型 Amazon CloudFormation 及 Amazon SAM 支持的参考信息，请参阅Amazon CloudFormation 用户指南中的Amazon 资源和属性类型参考。

在 Amazon SAM 模板中设置和管理资源访问权限

为了使您的 Amazon 资源相互交互，必须在您的资源之间配置适当的访问权限和权限。为此，需要配置 Amazon Identity and Access Management (IAM) 用户、角色和策略，以安全的方式完成互动。

本节中的主题都与设置对模板中定义的资源的访问相关。本节从一般最佳实践开始。接下来的两个主题回顾了在无服务器应用程序中引用的资源之间设置访问权限和权限的两个选项： Amazon SAM 连接器和 Amazon SAM 策略模板。最后一个主题详细介绍了如何使用与管理用户相同的机制 Amazon CloudFormation 来管理用户访问权限。

要了解更多信息，请参阅《Amazon CloudFormation 用户指南》中的使用 Amazon Identity and Access Management控制访问。

Amazon Serverless Application Model (Amazon SAM) 提供了两个选项，可简化对无服务器应用程序的访问和权限的管理。

1. Amazon SAM 连接器
2. Amazon SAM 策略模板

Amazon SAM 连接器

连接器是在两个资源之间配置权限的一种方式。为此，您可以通过在 Amazon SAM 模板中描述它们应如何相互交互来实现。可以使用 `Connectors` 资源属性或 `AWS::Serverless::Connector` 资源类型进行定义。连接器支持在资源组合之间配置`Read`和`Write`访问数据和事件。 Amazon 要了解有关 Amazon SAM 连接器的更多信息，请参阅使用 Amazon SAM 连接器管理资源权限。

Amazon SAM 策略模板

Amazon SAM 策略模板是预定义的权限集，您可以将其添加到 Amazon SAM 模板中，以管理您的 Amazon Lambda 函数、 Amazon Step Functions 状态机及其与之交互的资源之间的访问权限和权限。要了解有关 Amazon SAM 策略模板的更多信息，请参阅Amazon SAM策略模板。

Amazon CloudFormation 机制

Amazon CloudFormation 机制包括配置 IAM 用户、角色和策略，以管理您的 Amazon 资源之间的权限。要了解更多信息，请参阅 使用 Amazon CloudFormation 机制管理 Amazon SAM 权限。

最佳实践

在整个无服务器应用程序中，您可以使用多种方法来配置资源之间的权限。因此，您可以为每个场景选择最佳选项，并在整个应用程序中结合使用多个选项。选择最适合您的选项时，需要考虑以下几点：

- Amazon SAM 连接器和策略模板都减少了促进 Amazon 资源之间安全交互所需的 IAM 专业知识。如果受支持，请使用连接器和策略模板。

- Amazon SAM 连接器提供了一种简单直观的简短语法，用于在 Amazon SAM 模板中定义权限，并且只需要最少的 IAM 专业知识。当同时支持 Amazon SAM 连接器和策略模板时，请使用 Amazon SAM 连接器。
- Amazon SAM 连接器可以在支持的 Amazon SAM 源资源和目标资源之间配置`Read`和`Write`访问数据和事件。有关受支持的资源的列表，请参阅 Amazon SAM 连接器参考。如果支持，请使用 Amazon SAM 连接器。
- 虽然 Amazon SAM 策略模板仅限于您的 Lambda 函数、Step Functions 状态机及其与之交互的 Amazon 资源之间的权限，但策略模板确实支持所有 CRUD 操作。如果支持并且有适用于您的场景的 Amazon SAM 策略模板，请使用 Amazon SAM 策略模板。有关可用策略模板的列表，请参阅 Amazon SAM策略模板。
- 对于所有其他场景，或者需要精细度时，请使用 Amazon CloudFormation 机制。

使用 Amazon SAM 连接器管理资源权限

连接器是一种 Amazon Serverless Application Model (Amazon SAM) 抽象资源类型，标识为`AWS::Serverless::Connector`，它在您的无服务器应用程序资源之间提供简单且范围明确的权限。

Amazon SAM 连接器的好处

通过在资源之间自动编写适当的访问策略，连接器使您能够编写无服务器应用程序并专注于应用程序架构，而无需在 Amazon 授权功能、策略语言和特定于服务的安全设置方面的专业知识。因此，对于可能不熟悉无服务器开发的开发人员或希望提高开发速度的经验丰富的开发人员来说，连接器十分有用。

使用 Amazon SAM 连接器

通过将 `Connectors` 资源属性嵌入到源资源中来使用它。然后，定义您的目标资源并描述数据或事件应如何在这些资源之间流动。 Amazon SAM 然后制定必要的访问策略，以促进所需的交互。

以下内容概述了此资源属性的编写方式：

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
...
Resources:
  <source-resource-logical-id>:
    Type: <resource-type>
    ...
    Connectors:
```

```
      <connector-name>:
        Properties:
          Destination:
            <properties-that-identify-destination-resource>
          Permissions:
            <permission-types-to-provision>
  ...
```

连接器的工作原理

> **Note**
>
> 本节介绍连接器如何在幕后提供必要的资源。使用连接器时，这种情况会自动发生。

首先，嵌入的 `Connectors` 资源属性转换为 `AWS::Serverless::Connector` 资源类型。其逻辑 ID 会自动创建为*<source-resource-logical-id><embedded-connector-logical-id>*。

例如，这是嵌入式连接器：

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
...
Resources:
  MyFunction:
    Type: AWS::Lambda::Function
    Connectors:
      MyConn:
        Properties:
          Destination:
            Id: MyTable
          Permissions:
            - Read
            - Write
  MyTable:
    Type: AWS::DynamoDB::Table
```

这将生成以下 `AWS::Serverless::Connector` 资源：

```
Transform: AWS::Serverless-2016-10-31
Resources:
  ...
```

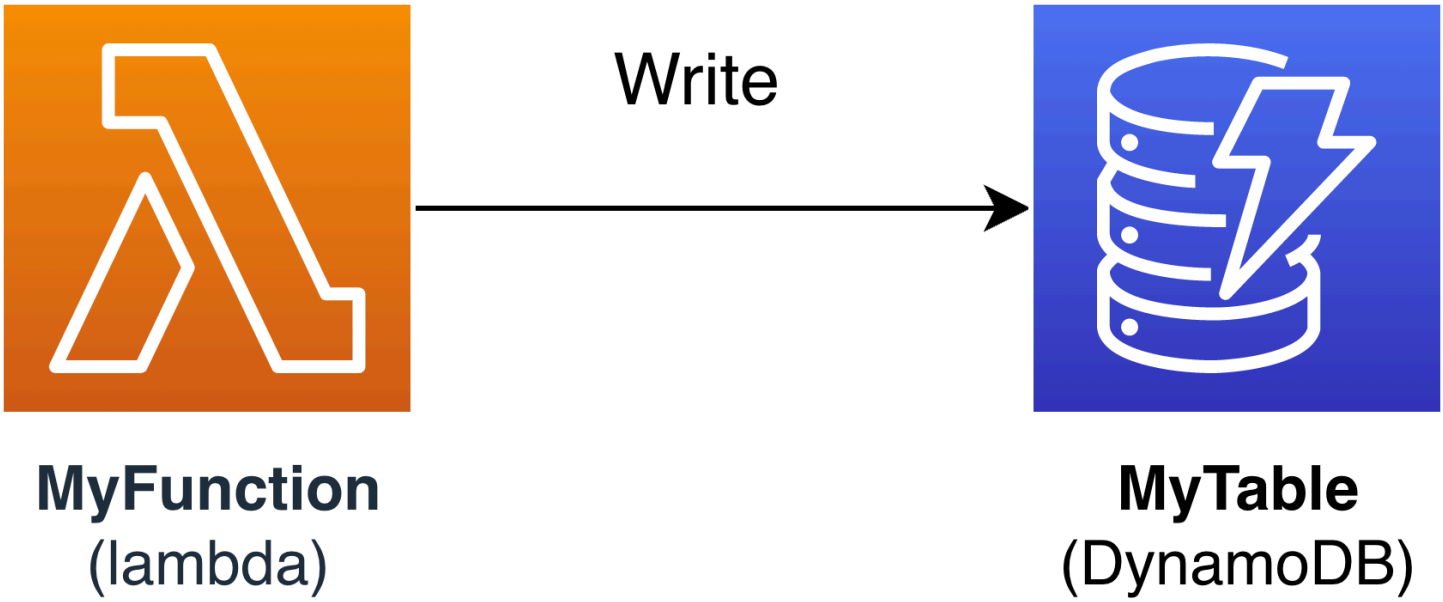
```
  MyFunctionMyConn:
    Type: AWS::Serverless::Connector
    Properties:
      Source:
        Id: MyFunction
      Destination:
        Id: MyTable
      Permissions:
        - Read
        - Write
```

> **Note**
>
> 您也可以使用此语法在 Amazon SAM 模板中定义连接器。如果源资源定义于与连接器不同的模板上，则建议使用此方法。

接下来，将自动编写该连接必要的访问策略。有关连接器生成的资源的更多信息，请参阅 Amazon CloudFormation 指定时生成的资源 AWS::Serverless::Connector。

连接器示例

以下示例说明如何使用连接器将数据从 Amazon Lambda 函数写入 Amazon DynamoDB 表。




```
Transform: AWS::Serverless-2016-10-31
Resources:
  MyTable:
```

```
    Type: AWS::Serverless::SimpleTable
  MyFunction:
    Type: AWS::Serverless::Function
    Connectors:
      MyConn:
        Properties:
          Destination:
            Id: MyTable
          Permissions:
            - Write
    Properties:
      Runtime: nodejs16.x
      Handler: index.handler
      InlineCode: |
        const AWS = require("aws-sdk");
        const docClient = new AWS.DynamoDB.DocumentClient();
        exports.handler = async (event, context) => {
          await docClient.put({
            TableName: process.env.TABLE_NAME,
            Item: {
              id: context.awsRequestId,
              event: JSON.stringify(event)
            }
          }).promise();
        }
      Environment:
        Variables:
          TABLE_NAME: !Ref MyTable
```

`Connectors` 资源属性嵌入在 Lambda 函数源资源中。使用 `Id` 属性将 DynamoDB 表定义为目标资源。连接器将在这两个资源之间提供 `Write` 权限。

当您将 Amazon SAM 模板部署到时 Amazon CloudFormation， Amazon SAM 将自动编写此连接正常运行所需的必要访问策略。

源资源和目标资源之间支持的连接

连接器支持源和目标资源连接的精选组合之间的 `Read` 和 `Write` 数据和事件权限类型。例如，连接器支持 `AWS::ApiGateway::RestApi` 源资源和 `AWS::Lambda::Function` 目标资源之间的 `Write` 连接。

可以使用支持的属性的组合来定义源资源和目标资源。属性要求将取决于您正在建立的连接以及资源定义的位置。

> **Note**
>
> 连接器可以在支持的无服务器和非无服务器资源类型之间配置权限。

有关支持的资源连接及其属性要求的列表，请参阅 连接器支持的源资源和目的地资源类型。

在中定义读取和写入权限 Amazon SAM

在Read和中 Amazon SAM，可以在单个连接器中配置Write权限：

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
...
Resources:
  MyFunction:
    Type: AWS::Lambda::Function
    Connectors:
      MyTableConn:
        Properties:
          Destination:
            Id: MyTable
          Permissions:
            - Read
            - Write
  MyTable:
    Type: AWS::DynamoDB::Table
```

有关使用连接器的更多信息，请参阅 Amazon SAM 连接器参考。

使用中其他支持的属性来定义资源 Amazon SAM

对于源资源和目标资源，如果在同一个模板中定义，则使用 Id 属性。或者，可以添加 Qualifier 以缩小您定义的资源范围。当资源不在同一个模板中时，请使用受支持属性的组合。

- 有关源资源和目标资源支持的属性组合列表，请参阅 连接器支持的源资源和目的地资源类型。
- 有关可用于连接器的属性的说明，请参见 AWS::Serverless::Connector。

在定义具有 Id 之外属性的源资源时，请使用 SourceReference 属性。

```
AWSTemplateFormatVersion: '2010-09-09'
```

```
Transform: AWS::Serverless-2016-10-31
...
Resources:
  <source-resource-logical-id>:
    Type: <resource-type>
    ...
    Connectors:
      <connector-name>:
        Properties:
          SourceReference:
            Qualifier: <optional-qualifier>
            <other-supported-properties>
          Destination:
            <properties-that-identify-destination-resource>
          Permissions:
            <permission-types-to-provision>
```

以下示例使用 `Qualifier` 缩小 Amazon API Gateway 资源的范围：

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
...
Resources:
  MyApi:
    Type: AWS::Serverless::Api
    Connectors:
      ApiToLambdaConn:
        Properties:
          SourceReference:
            Qualifier: Prod/GET/foobar
          Destination:
            Id: MyFunction
          Permissions:
            - Write
  ...
```

以下示例使用支持的 `Arn` 和 `Type` 组合定义来自另一个模板的目标资源：

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
...
Resources:
  MyFunction:
```

```
    Type: AWS::Serverless::Function
    Connectors:
      TableConn:
        Properties:
          Destination:
            Type: AWS::DynamoDB::Table
            Arn: !GetAtt MyTable.Arn
  ...
```

有关使用连接器的更多信息，请参阅 Amazon SAM 连接器参考。

从单一来源创建多个连接器 Amazon SAM

在一个源资源中，您可以定义多个连接器，且每个都有不同的目标资源。

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
...
Resources:
  MyFunction:
    Type: AWS::Serverless::Function
    Connectors:
      BucketConn:
        Properties:
          Destination:
            Id: amzn-s3-demo-bucket
          Permissions:
            - Read
            - Write
      SQSConn:
        Properties:
          Destination:
            Id: MyQueue
          Permissions:
            - Read
            - Write
      TableConn:
        Properties:
          Destination:
            Id: MyTable
          Permissions:
            - Read
            - Write
      TableConnWithTableArn:
```

```
        Properties:
          Destination:
            Type: AWS::DynamoDB::Table
            Arn: !GetAtt MyTable.Arn
          Permissions:
            - Read
            - Write
...
```

有关使用连接器的更多信息，请参阅 Amazon SAM 连接器参考。

在中创建多目的地连接器 Amazon SAM

在一个源资源中，您可以定义具有多个目标资源的单个连接器。以下是 Lambda 函数源资源的示例，其连接到 Amazon Simple Storage Service（Amazon S3）存储桶和 DynamoDB 表：

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
...
Resources:
  MyFunction:
    Type: AWS::Serverless::Function
    Connectors:
      WriteAccessConn:
        Properties:
          Destination:
            - Id: OutputBucket
            - Id: CredentialTable
          Permissions:
            - Write
  ...
  OutputBucket:
    Type: AWS::S3::Bucket
  CredentialTable:
    Type: AWS::DynamoDB::Table
```

有关使用连接器的更多信息，请参阅 Amazon SAM 连接器参考。

使用连接器定义资源属性 Amazon SAM

可以为资源定义资源属性，以指定其他行为和关系。要了解有关资源属性的更多信息，请参阅《Amazon CloudFormation 用户指南》中的资源属性参考。

您可以将资源属性添加到嵌入式连接器，方法是在与连接器属性相同的级别上对其进行定义。在部署时转换 Amazon SAM 模板时，属性将传递给生成的资源。

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
...
Resources:
  MyFunction:
    Type: AWS::Serverless::Function
    Connectors:
      MyConn:
        DeletionPolicy: Retain
        DependsOn: AnotherFunction
        Properties:
          ...
```

有关使用连接器的更多信息，请参阅 Amazon SAM 连接器参考。

了解更多

有关使用 Amazon SAM 连接器的更多信息，请参阅以下主题：

- AWS::Serverless::Connector
- 在中定义读取和写入权限 Amazon SAM
- 使用中其他支持的属性来定义资源 Amazon SAM
- 从单一来源创建多个连接器 Amazon SAM
- 在中创建多目的地连接器 Amazon SAM
- 在中定义读取和写入权限 Amazon SAM
- 使用连接器定义资源属性 Amazon SAM

提供反馈

要提供有关连接器的反馈，请在serverless-application-model Amazon GitHub存储库中提交新问题。

Amazon SAM策略模板

Amazon Serverless Application Model (Amazon SAM) 允许您从策略模板列表中进行选择，将 Lambda 函数和 Amazon Step Functions 状态机的权限范围限定为应用程序使用的资源。

Amazon SAM 中使用策略模板 Amazon Serverless Application Repository 的应用程序不需要任何特殊的客户确认即可从中部署应用程序。 Amazon Serverless Application Repository

如果要请求添加新的策略模板，请执行以下操作：

1. 针对项目分支中的 policy_templates.json 源文件提交拉取请求。`develop` Amazon SAM GitHub 你可以在网站上的 p olicy_templates. json 中找到源文件。 GitHub
2. 在 Amazon SAM GitHub 项目中提交一个问题，其中包含您的拉取请求的原因和请求链接。使用此链接提交新问题：Amazon Serverless Application Model：问题。

语法

对于您在模板文件中指定的每个策略 Amazon SAM 模板，必须始终指定一个包含策略模板占位符值的对象。如果策略模板不需要任何占位符值，则必须指定一个空对象。

YAML

```
MyFunction:
  Type: AWS::Serverless::Function
  Properties:
    Policies:
      - PolicyTemplateName1:        # Policy template with placeholder value
          Key1: Value1
      - PolicyTemplateName2: {}     # Policy template with no placeholder value
```

> **Note**
>
> 如果您已设置常规 IAM 策略或通过 Lambda 管理策略，则无需使用空对象即可设置策略模板。

示例

示例 1：具有占位符值的策略模板

以下示例显示 SQSPollerPolicy 策略模板期待 `QueueName` 作为资源。 Amazon SAM 模板检索 “`MyQueue`” Amazon SQS 队列的名称，您可以在同一个应用程序中创建该队列或请求该队列作为应用程序的参数。

```
MyFunction:
  Type: 'AWS::Serverless::Function'
  Properties:
```

```
    CodeUri: ${codeuri}
    Handler: hello.handler
    Runtime: python2.7
    Policies:
      - SQSPollerPolicy:
          QueueName:
            !GetAtt MyQueue.QueueName
```

示例 2：不具有占位符值的策略模板

以下示例包含 CloudWatchPutMetricPolicy 策略模板，该模板没有占位符值。

> **Note**
>
> 即使没有占位符值，也必须指定一个空对象，否则会导致错误。

```
MyFunction:
  Type: 'AWS::Serverless::Function'
  Properties:
    CodeUri: ${codeuri}
    Handler: hello.handler
    Runtime: python2.7
    Policies:
      - CloudWatchPutMetricPolicy: {}
```

示例 3：包含占位符值和常规 IAM 策略的策略模板

以下示例包含 Amazon A SQSFull cess 策略和DynamoDBCrudPolicy策略模板。Amazon A SQSFull ccess Amazon SAM 策略是一项 IAM 策略而不是策略，因此您不必指定空对象，因为该策略将直接传递给 Amazon CloudFormation。

```
MyFunction:
  Type: 'AWS::Serverless::Function'
  Properties:
    CodeUri: ${codeuri}
    Handler: hello.handler
    Runtime: python2.7
    Policies:
      - AmazonSQSFullAccess // IAM policy could be set without passing an empty object
```

```
      - DynamoDBCrudPolicy: // SAM specific policy, has a defined structure
          TableName:
            !Ref SampleTable
```

策略模板表

下表列出了可用的策略模板。

策略模板	描述
AcmGetCertificatePolicy	授予读取证书的权限 Amazon Certificate Manager。
AMIDescribePolicy	授予描述 Amazon 机器映像的权限 (AMIs)。
AthenaQueryPolicy	授予执行 Athena 查询的权限。
AWSSecretsManagerGetSecretValuePolicy	授予获取指定 Amazon Secrets Manager 密钥的密钥值的权限。
AWSSecretsManagerRotationPolicy	授予在 Amazon Secrets Manager中轮换密钥的权限。
CloudFormationDescribeStacksPolicy	授予描述 Amazon CloudFormation 堆栈的权限。
CloudWatchDashboardPolicy	授予在 CloudWatch 仪表板上放置要操作的指标的权限。
CloudWatchDescribe	授予描述 CloudWatch 警报历史记录的权限。

策略模板	描述		
AlarmHistoryPolicy			
CloudWatchPutMetricPolicy	授予向发送指标的权限 CloudWatch。		
CodeCommitCrudPolicy	为特定 CodeCommit存储库中的create/read/update/delete对象授予权限。		
CodeCommitReadPolicy	授予读取特定 CodeCommit 存储库中对象的权限。		
CodePipelineLambdaExecutionPolicy	允许调用的 Lambda 函数报告任务状态。CodePipeline		
CodePipelineReadOnlyPolicy	授予读取权限以获取有关 CodePipeline 管道的详细信息。		
ComprehendBasicAccessPolicy	授予检测实体、关键短语、语言和情绪的权限。		
CostExplorerReadOnlyPolicy	向只读的 Cost Explorer 授予账单历史记录 APIs 的只读权限。		
DynamoDBBackupFullAccessPolicy	授予对表进行 DynamoDB 按需备份的读和写权限。		
DynamoDBCrudPolicy	授予对 Amazon DynamoDB 表的创建、读取、更新和删除权限。		

策略模板	描述		
DynamoDBReadPolicy	授予对 DynamoDB 表的只读权限。		
DynamoDBReconfigurePolicy	授予重新配置 DynamoDB 表的权限。		
DynamoDBRestoreFromBackupPolicy	授予从备份还原 DynamoDB 表的权限。		
DynamoDBStreamReadPolicy	授予描述和读取 DynamoDB 流和记录的权限。		
DynamoDBWritePolicy	授予对 DynamoDB 表的只写权限。		
EC2CopyImagePolicy	允许复制 Amazon EC2 图片。		
EC2DescribePolicy	允许描述亚马逊弹性计算云 (Amazon EC2) 实例。		
EcsRunTaskPolicy	授予根据任务定义启动新任务的权限。		
EFSWriteAccessPolicy	授予挂载具有写入访问权限的 Amazon EFS 文件系统的权限。		
EKSDescribePolicy	授予描述或列出 Amazon EKS 集群的权限。		
ElasticMapReduceAddJobFlowStepsPolicy	授予将新步骤添加到运行的集群中的权限。		

策略模板	描述		
ElasticMapReduceCancelStepsPolicy	授予取消正在运行中的集群中的一个或多个待处理步骤的权限。		
ElasticMapReduceModifyInstanceFleetPolicy	授予列出集群内实例集的详细信息和修改这些实例集的容量的权限。		
ElasticMapReduceModifyInstanceGroupsPolicy	授予列出集群内实例组的详细信息和修改这些实例组的设置的权限。		
ElasticMapReduceSetTerminationProtectionPolicy	授予为集群设置终止保护的权限。		
ElasticMapReduceTerminateJobFlowsPolicy	授予关闭集群的权限。		
ElasticsearchHttpPostPolicy	向 Amazon OpenSearch 服务授予 POST 权限。		
EventBridgePutEventsPolicy	授予向发送事件的权限 EventBridge。		

策略模板	描述		
FilterLogEventsPolicy	授予筛选指定 CloudWatch 日志组中的日志事件的权限。		
FirehoseCrudPolicy	授予创建、写入、更新和删除 Firehose 传输流的权限。		
FirehoseWritePolicy	授予写入 Firehose 传输流的权限。		
KinesisCrudPolicy	授予创建、发布和删除 Amazon Kinesis 流的权限。		
KinesisStreamReadPolicy	授予列出和读取 Amazon Kinesis 流的权限。		
KMSDecryptPolicy	授予使用 Amazon Key Management Service (Amazon KMS) 密钥解密的权限。		
KMSEncryptPolicy	允许使用 Amazon Key Management Service (Amazon KMS) 密钥进行加密。		
LambdaInvokePolicy	授予调用 Amazon Lambda 函数、别名或版本的权限。		
MobileAnalyticsWriteOnlyAccessPolicy	授予对所有应用程序资源放置事件数据的只写权限。		
OrganizationsListAccountsPolicy	授予列出子女账户名称的只读权限，以及 IDs.		
PinpointEndpointAccessPolicy	授予为 Amazon Pinpoint 应用程序获取并更新端点的权限。		

策略模板	描述		
PollyFullAccessPolicy	授予对 Amazon Polly 词典资源的完全访问权限。		
RekognitionDetectOnlyPolicy	授予检测人脸、标签和文本的权限。		
RekognitionFacesManagementPolicy	授予在 Amazon Rekognition 集合中添加、删除和搜索人脸的权限。		
RekognitionFacesPolicy	授予比较并检测面部和标签的权限。		
RekognitionLabelsPolicy	授予检测对象和审核标签的权限。		
RekognitionNoDataAccessPolicy	授予比较并检测面部和标签的权限。		
RekognitionReadPolicy	授予列出和搜索人脸的权限。		
RekognitionWriteOnlyAccessPolicy	授予创建集合和为人脸编制索引的权限。		
Route53ChangeResourceRecordSetsPolicy	授予更改 Route 53 中的资源记录集的权限。		
S3CrudPolicy	授予创建、读取、更新和删除权限，以便对 Amazon S3 存储桶中的对象执行操作。		

策略模板	描述
S3FullAccessPolicy	授予完全访问权限，以便对 Amazon S3 存储桶中的对象执行操作。
S3ReadPolicy	授予读取 Amazon Simple Storage Service (Amazon S3) 存储桶中的对象的只读权限。
S3WritePolicy	授予将对象写入到 Amazon S3 存储桶的写入权限。
SageMakerCreateEndpointConfigPolicy	授予在 SageMaker AI 中创建终端节点配置的权限。
SageMakerCreateEndpointPolicy	授予在 SageMaker AI 中创建终端节点的权限。
ServerlessRepoReadWriteAccessPolicy	授予在 Amazon Serverless Application Repository 服务中创建和列出应用程序的权限。
SESBulkTemplatedCrudPolicy	授予发送电子邮件、模板化电子邮件和模板化批量电子邮件以及验证身份的权限。
SESBulkTemplatedCrudPolicy_v2	授予发送 Amazon SES 电子邮件、模板化电子邮件和模板化批量电子邮件以及验证身份的权限。
SESCrudPolicy	授予发送电子邮件和验证身份的权限。
SESEmailTemplateCrudPolicy	授予创建、获取、列出、更新和删除 Amazon SES 电子邮件模板的权限。

策略模板	描述		
SESSendBouncePolicy	SendBounce 授予亚马逊简单电子邮件服务 (Amazon SES) Service 身份的权限。		
SNSCrudPolicy	授予创建、发布和订阅 Amazon SNS 主题的权限。		
SNSPublishMessagePolicy	授予将消息发布到 Amazon Simple Notification Service (Amazon SNS)主题的权限。		
SQSPollerPolicy	授予轮询 Amazon Simple Queue Service (Amazon SQS) 队列的权限。		
SQSSendMessagePolicy	授予向 Amazon SQS 队列发送消息的权限。		
SSMParameterReadPolicy	授予访问来自 Amazon S EC2 ystems Manager (SSM) 参数存储库的参数的权限，以便在此账户中加载密钥。在参数名称不包含斜杠前缀时使用。		
SSMParameterWithSlashPrefixReadPolicy	授予访问来自 Amazon S EC2 ystems Manager (SSM) 参数存储库的参数的权限，以便在此账户中加载密钥。在参数名称包含斜杠前缀时使用。		
StepFunctionsExecutionPolicy	授予开始执行 Step Functions 状态机的权限。		
TextractDetectAnalyzePolicy	授予使用 Amazon Textract 检测和分析文档的权限。		
TextractGetResultPolicy	授予从 Amazon Textract 中获取检测到和分析过的文档的权限。		

策略模板	描述		
TextractPolicy	授予对 Amazon Textract 的完全访问权限。		
VPCAccessPolicy	授予访问权限以创建、删除、描述和分离弹性网络接口。		

故障排除

SAM CLI 错误：“必须为策略模板'< policy-template-name >'指定有效的参数值”

执行 `sam build` 时，您会看到以下错误：

```
"Must specify valid parameter values for policy template '<policy-template-name>'"
```

这意味着您在声明没有任何占位符值的策略模板时没有传递空对象。

要解决此问题，请如以下示例所示声明策略：CloudWatchPutMetricPolicy。

```
MyFunction:
  Policies:
    - CloudWatchPutMetricPolicy: {}
```

Amazon SAM 策略模板列表

以下是可用的策略模板以及应用于每个模板的权限。 Amazon Serverless Application Model (Amazon SAM) 会自动使用相应信息填充占位符项目（例如 Amazon 区域和账户 ID）。

主题

- AcmGetCertificatePolicy
- AMIDescribePolicy
- AthenaQueryPolicy
- AWSSecretsManagerGetSecretValuePolicy
- AWSSecretsManagerRotationPolicy
- CloudFormationDescribeStacksPolicy
- CloudWatchDashboardPolicy

- CloudWatchDescribeAlarmHistoryPolicy
- CloudWatchPutMetricPolicy
- CodePipelineLambdaExecutionPolicy
- CodePipelineReadOnlyPolicy
- CodeCommitCrudPolicy
- CodeCommitReadPolicy
- ComprehendBasicAccessPolicy
- CostExplorerReadOnlyPolicy
- DynamoDBBackupFullAccessPolicy
- DynamoDBCrudPolicy
- DynamoDBReadPolicy
- DynamoDBReconfigurePolicy
- DynamoDBRestoreFromBackupPolicy
- DynamoDBStreamReadPolicy
- DynamoDBWritePolicy
- EC2CopyImagePolicy
- EC2DescribePolicy
- EcsRunTaskPolicy
- EFSWriteAccessPolicy
- EKSDescribePolicy
- ElasticMapReduceAddJobFlowStepsPolicy
- ElasticMapReduceCancelStepsPolicy
- ElasticMapReduceModifyInstanceFleetPolicy
- ElasticMapReduceModifyInstanceGroupsPolicy
- ElasticMapReduceSetTerminationProtectionPolicy
- ElasticMapReduceTerminateJobFlowsPolicy
- ElasticsearchHttpPostPolicy
- EventBridgePutEventsPolicy
- FilterLogEventsPolicy
- FirehoseCrudPolicy

- FirehoseWritePolicy
- KinesisCrudPolicy
- KinesisStreamReadPolicy
- KMSDecryptPolicy
- KMSEncryptPolicy
- LambdaInvokePolicy
- MobileAnalyticsWriteOnlyAccessPolicy
- OrganizationsListAccountsPolicy
- PinpointEndpointAccessPolicy
- PollyFullAccessPolicy
- RekognitionDetectOnlyPolicy
- RekognitionFacesManagementPolicy
- RekognitionFacesPolicy
- RekognitionLabelsPolicy
- RekognitionNoDataAccessPolicy
- RekognitionReadPolicy
- RekognitionWriteOnlyAccessPolicy
- Route53ChangeResourceRecordSetsPolicy
- S3CrudPolicy
- S3FullAccessPolicy
- S3ReadPolicy
- S3WritePolicy
- SageMakerCreateEndpointConfigPolicy
- SageMakerCreateEndpointPolicy
- ServerlessRepoReadWriteAccessPolicy
- SESBulkTemplatedCrudPolicy
- SESBulkTemplatedCrudPolicy_v2
- SESCrudPolicy
- SESEmailTemplateCrudPolicy
- SESSendBouncePolicy

- SNSCrudPolicy
- SNSPublishMessagePolicy
- SQSPollerPolicy
- SQSSendMessagePolicy
- SSMParameterReadPolicy
- SSMParameterWithSlashPrefixReadPolicy
- StepFunctionsExecutionPolicy
- TextractDetectAnalyzePolicy
- TextractGetResultPolicy
- TextractPolicy
- VPCAccessPolicy

AcmGetCertificatePolicy

授予读取证书的权限 Amazon Certificate Manager。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "acm:GetCertificate"
    ],
    "Resource": {
      "Fn::Sub": [
        "${certificateArn}",
        {
          "certificateArn": {
            "Ref": "CertificateArn"
          }
        }
      ]
    }
  }
]
```

AMIDescribePolicy

授予描述 Amazon 机器映像的权限 (AMIs)。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "ec2:DescribeImages"
    ],
    "Resource": "*"
  }
]
```

AthenaQueryPolicy

授予执行 Athena 查询的权限。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "athena:ListWorkGroups",
      "athena:GetExecutionEngine",
      "athena:GetExecutionEngines",
      "athena:GetNamespace",
      "athena:GetCatalogs",
      "athena:GetNamespaces",
      "athena:GetTables",
      "athena:GetTable"
    ],
    "Resource": "*"
  },
  {
    "Effect": "Allow",
    "Action": [
      "athena:StartQueryExecution",
      "athena:GetQueryResults",
      "athena:DeleteNamedQuery",
      "athena:GetNamedQuery",
      "athena:ListQueryExecutions",
      "athena:StopQueryExecution",
      "athena:GetQueryResultsStream",
      "athena:ListNamedQueries",
      "athena:CreateNamedQuery",
      "athena:GetQueryExecution",
      "athena:BatchGetNamedQuery",
```

```
      "athena:BatchGetQueryExecution",
      "athena:GetWorkGroup"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:athena:${AWS::Region}:${AWS::AccountId}:workgroup/
${workgroupName}",
        {
          "workgroupName": {
            "Ref": "WorkGroupName"
          }
        }
      ]
    }
  }
]
```

AWSSecretsManagerGetSecretValuePolicy

授予获取指定 Amazon Secrets Manager 密钥的密钥值的权限。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "secretsmanager:GetSecretValue"
    ],
    "Resource": {
      "Fn::Sub": [
        "${secretArn}",
        {
          "secretArn": {
            "Ref": "SecretArn"
          }
        }
      ]
    }
  }
]
```

AWSSecretsManagerRotationPolicy

授予在 Amazon Secrets Manager中轮换密钥的权限。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "secretsmanager:DescribeSecret",
      "secretsmanager:GetSecretValue",
      "secretsmanager:PutSecretValue",
      "secretsmanager:UpdateSecretVersionStage"
    ],
    "Resource": {
      "Fn::Sub": "arn:${AWS::Partition}:secretsmanager:${AWS::Region}:
${AWS::AccountId}:secret:*"
    },
    "Condition": {
      "StringEquals": {
        "secretsmanager:resource/AllowRotationLambdaArn": {
          "Fn::Sub": [
            "arn:${AWS::Partition}:lambda:${AWS::Region}:${AWS::AccountId}:function:
${functionName}",
            {
              "functionName": {
                "Ref": "FunctionName"
              }
            }
          ]
        }
      }
    }
  },
  {
    "Effect": "Allow",
    "Action": [
      "secretsmanager:GetRandomPassword"
    ],
    "Resource": "*"
  }
]
```

CloudFormationDescribeStacksPolicy

授予描述 Amazon CloudFormation 堆栈的权限。

```
"Statement": [
```

```
  {
    "Effect": "Allow",
    "Action": [
      "cloudformation:DescribeStacks"
    ],
    "Resource": {
      "Fn::Sub": "arn:${AWS::Partition}:cloudformation:${AWS::Region}:
${AWS::AccountId}:stack/*"
    }
  }
]
```

CloudWatchDashboardPolicy

授予在 CloudWatch 仪表板上放置要操作的指标的权限。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "cloudwatch:GetDashboard",
      "cloudwatch:ListDashboards",
      "cloudwatch:PutDashboard",
      "cloudwatch:ListMetrics"
    ],
    "Resource": "*"
  }
]
```

CloudWatchDescribeAlarmHistoryPolicy

授予描述亚马逊 CloudWatch 警报历史记录的权限。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "cloudwatch:DescribeAlarmHistory"
    ],
    "Resource": "*"
  }
]
```

CloudWatchPutMetricPolicy

授予向发送指标的权限 CloudWatch。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "cloudwatch:PutMetricData"
    ],
    "Resource": "*"
  }
]
```

CodePipelineLambdaExecutionPolicy

允许调用的 Lambda 函数报告任务状态。 Amazon CodePipeline

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "codepipeline:PutJobSuccessResult",
      "codepipeline:PutJobFailureResult"
    ],
    "Resource": "*"
  }
]
```

CodePipelineReadOnlyPolicy

授予读取权限以获取有关 CodePipeline 管道的详细信息。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "codepipeline:ListPipelineExecutions"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:codepipeline:${AWS::Region}:${AWS::AccountId}:
${pipelinename}",
```

```
        {
          "pipelinename": {
            "Ref": "PipelineName"
          }
        }
      ]
    }
  }
]
```

CodeCommitCrudPolicy

授予在特定 CodeCommit存储库中创建、读取、更新和删除对象的权限。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "codecommit:GitPull",
      "codecommit:GitPush",
      "codecommit:CreateBranch",
      "codecommit:DeleteBranch",
      "codecommit:GetBranch",
      "codecommit:ListBranches",
      "codecommit:MergeBranchesByFastForward",
      "codecommit:MergeBranchesBySquash",
      "codecommit:MergeBranchesByThreeWay",
      "codecommit:UpdateDefaultBranch",
      "codecommit:BatchDescribeMergeConflicts",
      "codecommit:CreateUnreferencedMergeCommit",
      "codecommit:DescribeMergeConflicts",
      "codecommit:GetMergeCommit",
      "codecommit:GetMergeOptions",
      "codecommit:BatchGetPullRequests",
      "codecommit:CreatePullRequest",
      "codecommit:DescribePullRequestEvents",
      "codecommit:GetCommentsForPullRequest",
      "codecommit:GetCommitsFromMergeBase",
      "codecommit:GetMergeConflicts",
      "codecommit:GetPullRequest",
      "codecommit:ListPullRequests",
      "codecommit:MergePullRequestByFastForward",
      "codecommit:MergePullRequestBySquash",
      "codecommit:MergePullRequestByThreeWay",
```

```
      "codecommit:PostCommentForPullRequest",
      "codecommit:UpdatePullRequestDescription",
      "codecommit:UpdatePullRequestStatus",
      "codecommit:UpdatePullRequestTitle",
      "codecommit:DeleteFile",
      "codecommit:GetBlob",
      "codecommit:GetFile",
      "codecommit:GetFolder",
      "codecommit:PutFile",
      "codecommit:DeleteCommentContent",
      "codecommit:GetComment",
      "codecommit:GetCommentsForComparedCommit",
      "codecommit:PostCommentForComparedCommit",
      "codecommit:PostCommentReply",
      "codecommit:UpdateComment",
      "codecommit:BatchGetCommits",
      "codecommit:CreateCommit",
      "codecommit:GetCommit",
      "codecommit:GetCommitHistory",
      "codecommit:GetDifferences",
      "codecommit:GetObjectIdentifier",
      "codecommit:GetReferences",
      "codecommit:GetTree",
      "codecommit:GetRepository",
      "codecommit:UpdateRepositoryDescription",
      "codecommit:ListTagsForResource",
      "codecommit:TagResource",
      "codecommit:UntagResource",
      "codecommit:GetRepositoryTriggers",
      "codecommit:PutRepositoryTriggers",
      "codecommit:TestRepositoryTriggers",
      "codecommit:GetBranch",
      "codecommit:GetCommit",
      "codecommit:UploadArchive",
      "codecommit:GetUploadArchiveStatus",
      "codecommit:CancelUploadArchive"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:codecommit:${AWS::Region}:${AWS::AccountId}:
${repositoryName}",
        {
          "repositoryName": {
            "Ref": "RepositoryName"
```

```
          }
        }
      ]
    }
  }
]
```

CodeCommitReadPolicy

授予读取特定 CodeCommit 存储库中对象的权限。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "codecommit:GitPull",
      "codecommit:GetBranch",
      "codecommit:ListBranches",
      "codecommit:BatchDescribeMergeConflicts",
      "codecommit:DescribeMergeConflicts",
      "codecommit:GetMergeCommit",
      "codecommit:GetMergeOptions",
      "codecommit:BatchGetPullRequests",
      "codecommit:DescribePullRequestEvents",
      "codecommit:GetCommentsForPullRequest",
      "codecommit:GetCommitsFromMergeBase",
      "codecommit:GetMergeConflicts",
      "codecommit:GetPullRequest",
      "codecommit:ListPullRequests",
      "codecommit:GetBlob",
      "codecommit:GetFile",
      "codecommit:GetFolder",
      "codecommit:GetComment",
      "codecommit:GetCommentsForComparedCommit",
      "codecommit:BatchGetCommits",
      "codecommit:GetCommit",
      "codecommit:GetCommitHistory",
      "codecommit:GetDifferences",
      "codecommit:GetObjectIdentifier",
      "codecommit:GetReferences",
      "codecommit:GetTree",
      "codecommit:GetRepository",
      "codecommit:ListTagsForResource",
      "codecommit:GetRepositoryTriggers",
```

```
      "codecommit:TestRepositoryTriggers",
      "codecommit:GetBranch",
      "codecommit:GetCommit",
      "codecommit:GetUploadArchiveStatus"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:codecommit:${AWS::Region}:${AWS::AccountId}:
${repositoryName}",
        {
          "repositoryName": {
            "Ref": "RepositoryName"
          }
        }
      ]
    }
  }
]
```

ComprehendBasicAccessPolicy

授予检测实体、关键短语、语言和情绪的权限。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "comprehend:BatchDetectKeyPhrases",
      "comprehend:DetectDominantLanguage",
      "comprehend:DetectEntities",
      "comprehend:BatchDetectEntities",
      "comprehend:DetectKeyPhrases",
      "comprehend:DetectSentiment",
      "comprehend:BatchDetectDominantLanguage",
      "comprehend:BatchDetectSentiment"
    ],
    "Resource": "*"
  }
]
```

CostExplorerReadOnlyPolicy

向只读 Amazon Cost Explorer（Cost Explorer）授予账单历史记录 APIs 的只读权限。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "ce:GetCostAndUsage",
      "ce:GetDimensionValues",
      "ce:GetReservationCoverage",
      "ce:GetReservationPurchaseRecommendation",
      "ce:GetReservationUtilization",
      "ce:GetTags"
    ],
    "Resource": "*"
  }
]
```

DynamoDBBackupFullAccessPolicy

授予对表进行 DynamoDB 按需备份的读和写权限。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "dynamodb:CreateBackup",
      "dynamodb:DescribeContinuousBackups"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:dynamodb:${AWS::Region}:${AWS::AccountId}:table/
${tableName}",
        {
          "tableName": {
            "Ref": "TableName"
          }
        }
      ]
    }
  },
  {
    "Effect": "Allow",
    "Action": [
      "dynamodb:DeleteBackup",
      "dynamodb:DescribeBackup",
```

```
      "dynamodb:ListBackups"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:dynamodb:${AWS::Region}:${AWS::AccountId}:table/
${tableName}/backup/*",
        {
          "tableName": {
            "Ref": "TableName"
          }
        }
      ]
    }
  }
]
```

DynamoDBCrudPolicy

授予对 Amazon DynamoDB 表的创建、读取、更新和删除权限。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "dynamodb:GetItem",
      "dynamodb:DeleteItem",
      "dynamodb:PutItem",
      "dynamodb:Scan",
      "dynamodb:Query",
      "dynamodb:UpdateItem",
      "dynamodb:BatchWriteItem",
      "dynamodb:BatchGetItem",
      "dynamodb:DescribeTable",
      "dynamodb:ConditionCheckItem"
    ],
    "Resource": [
      {
        "Fn::Sub": [
          "arn:${AWS::Partition}:dynamodb:${AWS::Region}:${AWS::AccountId}:table/
${tableName}",
          {
            "tableName": {
              "Ref": "TableName"
            }
```

```
          }
        ]
      },
      {
        "Fn::Sub": [
          "arn:${AWS::Partition}:dynamodb:${AWS::Region}:${AWS::AccountId}:table/
${tableName}/index/*",
          {
            "tableName": {
              "Ref": "TableName"
            }
          }
        ]
      }
    ]
  }
]
```

DynamoDBReadPolicy

授予对 DynamoDB 表的只读权限。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "dynamodb:GetItem",
      "dynamodb:Scan",
      "dynamodb:Query",
      "dynamodb:BatchGetItem",
      "dynamodb:DescribeTable"
    ],
    "Resource": [
      {
        "Fn::Sub": [
          "arn:${AWS::Partition}:dynamodb:${AWS::Region}:${AWS::AccountId}:table/
${tableName}",
          {
            "tableName": {
              "Ref": "TableName"
            }
          }
        ]
      },
```

```
      {
        "Fn::Sub": [
          "arn:${AWS::Partition}:dynamodb:${AWS::Region}:${AWS::AccountId}:table/
${tableName}/index/*",
          {
            "tableName": {
              "Ref": "TableName"
            }
          }
        ]
      }
    ]
  }
]
```

DynamoDBReconfigurePolicy

授予重新配置 DynamoDB 表的权限。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "dynamodb:UpdateTable"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:dynamodb:${AWS::Region}:${AWS::AccountId}:table/
${tableName}",
        {
          "tableName": {
            "Ref": "TableName"
          }
        }
      ]
    }
  }
]
```

DynamoDBRestoreFromBackupPolicy

授予从备份还原 DynamoDB 表的权限。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "dynamodb:RestoreTableFromBackup"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:dynamodb:${AWS::Region}:${AWS::AccountId}:table/
${tableName}/backup/*",
        {
          "tableName": {
            "Ref": "TableName"
          }
        }
      ]
    }
  },
  {
    "Effect": "Allow",
    "Action": [
      "dynamodb:PutItem",
      "dynamodb:UpdateItem",
      "dynamodb:DeleteItem",
      "dynamodb:GetItem",
      "dynamodb:Query",
      "dynamodb:Scan",
      "dynamodb:BatchWriteItem"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:dynamodb:${AWS::Region}:${AWS::AccountId}:table/
${tableName}",
        {
          "tableName": {
            "Ref": "TableName"
          }
        }
      ]
    }
  }
]
```

DynamoDBStreamReadPolicy

授予描述和读取 DynamoDB 流和记录的权限。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "dynamodb:DescribeStream",
      "dynamodb:GetRecords",
      "dynamodb:GetShardIterator"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:dynamodb:${AWS::Region}:${AWS::AccountId}:table/
${tableName}/stream/${streamName}",
        {
          "tableName": {
            "Ref": "TableName"
          },
          "streamName": {
            "Ref": "StreamName"
          }
        }
      ]
    }
  },
  {
    "Effect": "Allow",
    "Action": [
      "dynamodb:ListStreams"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:dynamodb:${AWS::Region}:${AWS::AccountId}:table/
${tableName}/stream/*",
        {
          "tableName": {
            "Ref": "TableName"
          }
        }
      ]
    }
  }
```

```
]
```

DynamoDBWritePolicy

授予对 DynamoDB 表的只写权限。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "dynamodb:PutItem",
      "dynamodb:UpdateItem",
      "dynamodb:BatchWriteItem"
    ],
    "Resource": [
      {
        "Fn::Sub": [
          "arn:${AWS::Partition}:dynamodb:${AWS::Region}:${AWS::AccountId}:table/
${tableName}",
          {
            "tableName": {
              "Ref": "TableName"
            }
          }
        ]
      },
      {
        "Fn::Sub": [
          "arn:${AWS::Partition}:dynamodb:${AWS::Region}:${AWS::AccountId}:table/
${tableName}/index/*",
          {
            "tableName": {
              "Ref": "TableName"
            }
          }
        ]
      }
    ]
  }
]
```

EC2CopyImagePolicy

允许复制 Amazon EC2 图片。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "ec2:CopyImage"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:ec2:${AWS::Region}:${AWS::AccountId}:image/${imageId}",
        {
          "imageId": {
            "Ref": "ImageId"
          }
        }
      ]
    }
  }
]
```

EC2DescribePolicy

允许描述亚马逊弹性计算云 (Amazon EC2) 实例。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "ec2:DescribeRegions",
      "ec2:DescribeInstances"
    ],
    "Resource": "*"
  }
]
```

EcsRunTaskPolicy

授予根据任务定义启动新任务的权限。

```
"Statement": [
```

```
  {
    "Action": [
      "ecs:RunTask"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:ecs:${AWS::Region}:${AWS::AccountId}:task-definition/
${taskDefinition}",
        {
          "taskDefinition": {
            "Ref": "TaskDefinition"
          }
        }
      ]
    },
    "Effect": "Allow"
  }
]
```

EFSWriteAccessPolicy

授予挂载具有写入访问权限的 Amazon EFS 文件系统的权限。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "elasticfilesystem:ClientMount",
      "elasticfilesystem:ClientWrite"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:elasticfilesystem:${AWS::Region}:${AWS::AccountId}:file-
system/${FileSystem}",
        {
          "FileSystem": {
            "Ref": "FileSystem"
          }
        }
      ]
    },
    "Condition": {
      "StringEquals": {
        "elasticfilesystem:AccessPointArn": {
```

```
          "Fn::Sub": [
            "arn:${AWS::Partition}:elasticfilesystem:${AWS::Region}:
${AWS::AccountId}:access-point/${AccessPoint}",
            {
              "AccessPoint": {
                "Ref": "AccessPoint"
              }
            }
          ]
        }
      }
    }
  }
]
```

EKSDescribePolicy

授予描述或列出 Amazon Elastic Kubernetes Service (Amazon EKS) 集群的权限。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "eks:DescribeCluster",
      "eks:ListClusters"
    ],
    "Resource": "*"
  }
]
```

ElasticMapReduceAddJobFlowStepsPolicy

授予将新步骤添加到运行的集群中的权限。

```
"Statement": [
  {
    "Action": "elasticmapreduce:AddJobFlowSteps",
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:elasticmapreduce:${AWS::Region}:
${AWS::AccountId}:cluster/${clusterId}",
        {
```

```
          "clusterId": {
            "Ref": "ClusterId"
          }
        }
      ]
    },
    "Effect": "Allow"
  }
]
```

ElasticMapReduceCancelStepsPolicy

授予取消正在运行中的集群中的一个或多个待处理步骤的权限。

```
"Statement": [
  {
    "Action": "elasticmapreduce:CancelSteps",
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:elasticmapreduce:${AWS::Region}:
${AWS::AccountId}:cluster/${clusterId}",
        {
          "clusterId": {
            "Ref": "ClusterId"
          }
        }
      ]
    },
    "Effect": "Allow"
  }
]
```

ElasticMapReduceModifyInstanceFleetPolicy

授予列出集群内实例集的详细信息和修改这些实例集的容量的权限。

```
"Statement": [
  {
    "Action": [
      "elasticmapreduce:ModifyInstanceFleet",
      "elasticmapreduce:ListInstanceFleets"
    ],
```

```
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:elasticmapreduce:${AWS::Region}:
${AWS::AccountId}:cluster/${clusterId}",
        {
          "clusterId": {
            "Ref": "ClusterId"
          }
        }
      ]
    },
    "Effect": "Allow"
  }
]
```

ElasticMapReduceModifyInstanceGroupsPolicy

授予列出集群内实例组的详细信息和修改这些实例组的设置的权限。

```
"Statement": [
  {
    "Action": [
      "elasticmapreduce:ModifyInstanceGroups",
      "elasticmapreduce:ListInstanceGroups"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:elasticmapreduce:${AWS::Region}:
${AWS::AccountId}:cluster/${clusterId}",
        {
          "clusterId": {
            "Ref": "ClusterId"
          }
        }
      ]
    },
    "Effect": "Allow"
  }
]
```

ElasticMapReduceSetTerminationProtectionPolicy

授予为集群设置终止保护的权限。

```
"Statement": [
  {
    "Action": "elasticmapreduce:SetTerminationProtection",
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:elasticmapreduce:${AWS::Region}:
${AWS::AccountId}:cluster/${clusterId}",
        {
          "clusterId": {
            "Ref": "ClusterId"
          }
        }
      ]
    },
    "Effect": "Allow"
  }
]
```

ElasticMapReduceTerminateJobFlowsPolicy

授予关闭集群的权限。

```
"Statement": [
  {
    "Action": "elasticmapreduce:TerminateJobFlows",
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:elasticmapreduce:${AWS::Region}:
${AWS::AccountId}:cluster/${clusterId}",
        {
          "clusterId": {
            "Ref": "ClusterId"
          }
        }
      ]
    },
    "Effect": "Allow"
  }
]
```

ElasticsearchHttpPostPolicy

向 Amazon OpenSearch 服务授予 POST 和 PUT 权限。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "es:ESHttpPost",
      "es:ESHttpPut"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:es:${AWS::Region}:${AWS::AccountId}:domain/
${domainName}/*",
        {
          "domainName": {
            "Ref": "DomainName"
          }
        }
      ]
    }
  }
]
```

EventBridgePutEventsPolicy

授予向 Amazon 发送事件的权限 EventBridge。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": "events:PutEvents",
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:events:${AWS::Region}:${AWS::AccountId}:event-bus/
${eventBusName}",
        {
          "eventBusName": {
            "Ref": "EventBusName"
          }
        }
      ]
    }
  }
]
```

FilterLogEventsPolicy

授予筛选指定 CloudWatch 日志组中的日志事件的权限。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "logs:FilterLogEvents"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:logs:${AWS::Region}:${AWS::AccountId}:log-group:
${logGroupName}:log-stream:*",
        {
          "logGroupName": {
            "Ref": "LogGroupName"
          }
        }
      ]
    }
  }
]
```

FirehoseCrudPolicy

授予创建、写入、更新和删除 Firehose 传输流的权限。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "firehose:CreateDeliveryStream",
      "firehose:DeleteDeliveryStream",
      "firehose:DescribeDeliveryStream",
      "firehose:PutRecord",
      "firehose:PutRecordBatch",
      "firehose:UpdateDestination"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:firehose:${AWS::Region}:
${AWS::AccountId}:deliverystream/${deliveryStreamName}",
```

```
        {
          "deliveryStreamName": {
            "Ref": "DeliveryStreamName"
          }
        }
      ]
    }
  }
]
```

FirehoseWritePolicy

授予写入 Firehose 传输流的权限。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "firehose:PutRecord",
      "firehose:PutRecordBatch"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:firehose:${AWS::Region}:
${AWS::AccountId}:deliverystream/${deliveryStreamName}",
        {
          "deliveryStreamName": {
            "Ref": "DeliveryStreamName"
          }
        }
      ]
    }
  }
]
```

KinesisCrudPolicy

授予创建、发布和删除 Amazon Kinesis 流的权限。

```
"Statement": [
  {
    "Effect": "Allow",
```

```
    "Action": [
      "kinesis:AddTagsToStream",
      "kinesis:CreateStream",
      "kinesis:DecreaseStreamRetentionPeriod",
      "kinesis:DeleteStream",
      "kinesis:DescribeStream",
      "kinesis:DescribeStreamSummary",
      "kinesis:GetShardIterator",
      "kinesis:IncreaseStreamRetentionPeriod",
      "kinesis:ListTagsForStream",
      "kinesis:MergeShards",
      "kinesis:PutRecord",
      "kinesis:PutRecords",
      "kinesis:SplitShard",
      "kinesis:RemoveTagsFromStream"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:kinesis:${AWS::Region}:${AWS::AccountId}:stream/
${streamName}",
        {
          "streamName": {
            "Ref": "StreamName"
          }
        }
      ]
    }
  }
]
```

KinesisStreamReadPolicy

授予列出和读取 Amazon Kinesis 流的权限。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "kinesis:ListStreams",
      "kinesis:DescribeLimits"
    ],
    "Resource": {
      "Fn::Sub": "arn:${AWS::Partition}:kinesis:${AWS::Region}:
${AWS::AccountId}:stream/*"
```

```
    }
  },
  {
    "Effect": "Allow",
    "Action": [
      "kinesis:DescribeStream",
      "kinesis:DescribeStreamSummary",
      "kinesis:GetRecords",
      "kinesis:GetShardIterator"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:kinesis:${AWS::Region}:${AWS::AccountId}:stream/
${streamName}",
        {
          "streamName": {
            "Ref": "StreamName"
          }
        }
      ]
    }
  }
]
```

KMSDecryptPolicy

授予使用 Amazon Key Management Service (Amazon KMS) 密钥解密的权限。请注意，`keyId`必须是 Amazon KMS 密钥 ID，而不是密钥别名。

```
"Statement": [
  {
    "Action": "kms:Decrypt",
    "Effect": "Allow",
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:kms:${AWS::Region}:${AWS::AccountId}:key/${keyId}",
        {
          "keyId": {
            "Ref": "KeyId"
          }
        }
      ]
    }
```

```
  }
]
```

KMSEncryptPolicy

允许使用密 Amazon KMS 钥进行加密。请注意，keyID 必须是 Amazon KMS 密钥 ID，而不是密钥别名。

```
"Statement": [
  {
    "Action": "kms:Encrypt",
    "Effect": "Allow",
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:kms:${AWS::Region}:${AWS::AccountId}:key/${keyId}",
        {
          "keyId": {
            "Ref": "KeyId"
          }
        }
      ]
    }
  }
]
```

LambdaInvokePolicy

授予调用 Amazon Lambda 函数、别名或版本的权限。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "lambda:InvokeFunction"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:lambda:${AWS::Region}:${AWS::AccountId}:function:
${functionName}*",
        {
          "functionName": {
            "Ref": "FunctionName"
```

```
          }
        }
      ]
    }
  }
]
```

MobileAnalyticsWriteOnlyAccessPolicy

授予对所有应用程序资源放置事件数据的只写权限。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "mobileanalytics:PutEvents"
    ],
    "Resource": "*"
  }
]
```

OrganizationsListAccountsPolicy

授予列出子女账户名称的只读权限，以及 IDs.

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "organizations:ListAccounts"
    ],
    "Resource": "*"
  }
]
```

PinpointEndpointAccessPolicy

授予为 Amazon Pinpoint 应用程序获取并更新端点的权限。

```
"Statement": [
  {
```

```
    "Effect": "Allow",
    "Action": [
      "mobiletargeting:GetEndpoint",
      "mobiletargeting:UpdateEndpoint",
      "mobiletargeting:UpdateEndpointsBatch"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:mobiletargeting:${AWS::Region}:${AWS::AccountId}:apps/
${pinpointApplicationId}/endpoints/*",
        {
          "pinpointApplicationId": {
            "Ref": "PinpointApplicationId"
          }
        }
      ]
    }
  }
]
```

PollyFullAccessPolicy

授予对 Amazon Polly 词典资源的完全访问权限。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "polly:GetLexicon",
      "polly:DeleteLexicon"
    ],
    "Resource": [
      {
        "Fn::Sub": [
          "arn:${AWS::Partition}:polly:${AWS::Region}:${AWS::AccountId}:lexicon/
${lexiconName}",
          {
            "lexiconName": {
              "Ref": "LexiconName"
            }
          }
        ]
      }
    ]
```

```
  },
  {
    "Effect": "Allow",
    "Action": [
      "polly:DescribeVoices",
      "polly:ListLexicons",
      "polly:PutLexicon",
      "polly:SynthesizeSpeech"
    ],
    "Resource": [
      {
        "Fn::Sub": "arn:${AWS::Partition}:polly:${AWS::Region}:
${AWS::AccountId}:lexicon/*"
      }
    ]
  }
]
```

RekognitionDetectOnlyPolicy

授予检测人脸、标签和文本的权限。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "rekognition:DetectFaces",
      "rekognition:DetectLabels",
      "rekognition:DetectModerationLabels",
      "rekognition:DetectText"
    ],
    "Resource": "*"
  }
]
```

RekognitionFacesManagementPolicy

授予在 Amazon Rekognition 集合中添加、删除和搜索人脸的权限。

```
"Statement": [
  {
    "Effect": "Allow",
```

```
    "Action": [
      "rekognition:IndexFaces",
      "rekognition:DeleteFaces",
      "rekognition:SearchFaces",
      "rekognition:SearchFacesByImage",
      "rekognition:ListFaces"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:rekognition:${AWS::Region}:${AWS::AccountId}:collection/
${collectionId}",
        {
          "collectionId": {
            "Ref": "CollectionId"
          }
        }
      ]
    }
  }
]
```

RekognitionFacesPolicy

授予比较并检测面部和标签的权限。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "rekognition:CompareFaces",
      "rekognition:DetectFaces"
    ],
    "Resource": "*"
  }
]
```

RekognitionLabelsPolicy

授予检测对象和审核标签的权限。

```
"Statement": [
  {
```

```
    "Effect": "Allow",
    "Action": [
      "rekognition:DetectLabels",
      "rekognition:DetectModerationLabels"
    ],
    "Resource": "*"
  }
]
```

RekognitionNoDataAccessPolicy

授予比较并检测面部和标签的权限。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "rekognition:CompareFaces",
      "rekognition:DetectFaces",
      "rekognition:DetectLabels",
      "rekognition:DetectModerationLabels"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:rekognition:${AWS::Region}:${AWS::AccountId}:collection/
${collectionId}",
        {
          "collectionId": {
            "Ref": "CollectionId"
          }
        }
      ]
    }
  }
]
```

RekognitionReadPolicy

授予列出和搜索人脸的权限。

```
"Statement": [
  {
```

```
    "Effect": "Allow",
    "Action": [
      "rekognition:ListCollections",
      "rekognition:ListFaces",
      "rekognition:SearchFaces",
      "rekognition:SearchFacesByImage"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:rekognition:${AWS::Region}:${AWS::AccountId}:collection/
${collectionId}",
        {
          "collectionId": {
            "Ref": "CollectionId"
          }
        }
      ]
    }
  }
]
```

RekognitionWriteOnlyAccessPolicy

授予创建集合和为人脸编制索引的权限。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "rekognition:CreateCollection",
      "rekognition:IndexFaces"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:rekognition:${AWS::Region}:${AWS::AccountId}:collection/
${collectionId}",
        {
          "collectionId": {
            "Ref": "CollectionId"
          }
        }
      ]
    }
  }
```

```
]
```

Route53ChangeResourceRecordSetsPolicy

授予更改 Route 53 中的资源记录集的权限。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "route53:ChangeResourceRecordSets"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:route53:::hostedzone/${HostedZoneId}",
        {
          "HostedZoneId": {
            "Ref": "HostedZoneId"
          }
        }
      ]
    }
  }
]
```

S3CrudPolicy

授予创建、读取、更新和删除权限，以便对 Amazon S3 存储桶中的对象执行操作。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "s3:GetObject",
      "s3:ListBucket",
      "s3:GetBucketLocation",
      "s3:GetObjectVersion",
      "s3:PutObject",
      "s3:PutObjectAcl",
      "s3:GetLifecycleConfiguration",
      "s3:PutLifecycleConfiguration",
      "s3:DeleteObject"
    ],
```

```
    "Resource": [
      {
        "Fn::Sub": [
          "arn:${AWS::Partition}:s3:::${bucketName}",
          {
            "bucketName": {
              "Ref": "BucketName"
            }
          }
        ]
      },
      {
        "Fn::Sub": [
          "arn:${AWS::Partition}:s3:::${bucketName}/*",
          {
            "bucketName": {
              "Ref": "BucketName"
            }
          }
        ]
      }
    ]
  }
]
```

S3FullAccessPolicy

授予完全访问权限，以便对 Amazon S3 存储桶中的对象执行操作。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "s3:GetObject",
      "s3:GetObjectAcl",
      "s3:GetObjectVersion",
      "s3:PutObject",
      "s3:PutObjectAcl",
      "s3:DeleteObject",
      "s3:DeleteObjectTagging",
      "s3:DeleteObjectVersionTagging",
      "s3:GetObjectTagging",
      "s3:GetObjectVersionTagging",
      "s3:PutObjectTagging",
```

```
      "s3:PutObjectVersionTagging"
    ],
    "Resource": [
      {
        "Fn::Sub": [
          "arn:${AWS::Partition}:s3:::${bucketName}/*",
          {
            "bucketName": {
              "Ref": "BucketName"
            }
          }
        ]
      }
    ]
  },
  {
    "Effect": "Allow",
    "Action": [
      "s3:ListBucket",
      "s3:GetBucketLocation",
      "s3:GetLifecycleConfiguration",
      "s3:PutLifecycleConfiguration"
    ],
    "Resource": [
      {
        "Fn::Sub": [
          "arn:${AWS::Partition}:s3:::${bucketName}",
          {
            "bucketName": {
              "Ref": "BucketName"
            }
          }
        ]
      }
    ]
  }
]
```

S3ReadPolicy

授予读取 Amazon Simple Storage Service (Amazon S3) 存储桶中的对象的只读权限。

```
"Statement": [
  {
```

```
    "Effect": "Allow",
    "Action": [
      "s3:GetObject",
      "s3:ListBucket",
      "s3:GetBucketLocation",
      "s3:GetObjectVersion",
      "s3:GetLifecycleConfiguration"
    ],
    "Resource": [
      {
        "Fn::Sub": [
          "arn:${AWS::Partition}:s3:::${bucketName}",
          {
            "bucketName": {
              "Ref": "BucketName"
            }
          }
        ]
      },
      {
        "Fn::Sub": [
          "arn:${AWS::Partition}:s3:::${bucketName}/*",
          {
            "bucketName": {
              "Ref": "BucketName"
            }
          }
        ]
      }
    ]
  }
]
```

S3WritePolicy

授予将对象写入到 Amazon S3 存储桶的写入权限。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "s3:PutObject",
      "s3:PutObjectAcl",
      "s3:PutLifecycleConfiguration"
```

```
    ],
    "Resource": [
      {
        "Fn::Sub": [
          "arn:${AWS::Partition}:s3:::${bucketName}",
          {
            "bucketName": {
              "Ref": "BucketName"
            }
          }
        ]
      },
      {
        "Fn::Sub": [
          "arn:${AWS::Partition}:s3:::${bucketName}/*",
          {
            "bucketName": {
              "Ref": "BucketName"
            }
          }
        ]
      }
    ]
  }
]
```

SageMakerCreateEndpointConfigPolicy

授予在 SageMaker AI 中创建终端节点配置的权限。

```
"Statement": [
  {
    "Action": [
      "sagemaker:CreateEndpointConfig"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:sagemaker:${AWS::Region}:${AWS::AccountId}:endpoint-
config/${endpointConfigName}",
        {
          "endpointConfigName": {
            "Ref": "EndpointConfigName"
          }
        }
```

```
      ]
    },
    "Effect": "Allow"
  }
]
```

SageMakerCreateEndpointPolicy

授予在 SageMaker AI 中创建终端节点的权限。

```
"Statement": [
  {
    "Action": [
      "sagemaker:CreateEndpoint"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:sagemaker:${AWS::Region}:${AWS::AccountId}:endpoint/
${endpointName}",
        {
          "endpointName": {
            "Ref": "EndpointName"
          }
        }
      ]
    },
    "Effect": "Allow"
  }
]
```

ServerlessRepoReadWriteAccessPolicy

授予在 Amazon Serverless Application Repository (Amazon SAM) 服务中创建和列出应用程序的权限。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "serverlessrepo:CreateApplication",
      "serverlessrepo:CreateApplicationVersion",
      "serverlessrepo:GetApplication",
```

```
      "serverlessrepo:ListApplications",
      "serverlessrepo:ListApplicationVersions"
    ],
    "Resource": [
      {
        "Fn::Sub": "arn:${AWS::Partition}:serverlessrepo:${AWS::Region}:
${AWS::AccountId}:applications/*"
      }
    ]
  }
]
```

SESBulkTemplatedCrudPolicy

授予发送 Amazon SES 电子邮件、模板化电子邮件和模板化批量电子邮件以及验证身份的权限。

> **Note**
>
> `ses:SendTemplatedEmail` 操作需要模板 ARN。请改用 `SESBulkTemplatedCrudPolicy_v2`。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "ses:GetIdentityVerificationAttributes",
      "ses:SendEmail",
      "ses:SendRawEmail",
      "ses:SendTemplatedEmail",
      "ses:SendBulkTemplatedEmail",
      "ses:VerifyEmailIdentity"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:ses:${AWS::Region}:${AWS::AccountId}:identity/
${identityName}",
        {
          "identityName": {
            "Ref": "IdentityName"
          }
        }
```

```
      ]
    }
  }
]
```

SESBulkTemplatedCrudPolicy_v2

授予发送 Amazon SES 电子邮件、模板化电子邮件和模板化批量电子邮件以及验证身份的权限。

```
"Statement": [
  {
    "Action": [
      "ses:SendEmail",
      "ses:SendRawEmail",
      "ses:SendTemplatedEmail",
      "ses:SendBulkTemplatedEmail"
    ],
    "Effect": "Allow",
    "Resource": [
      {
        "Fn::Sub": [
          "arn:${AWS::Partition}:ses:${AWS::Region}:${AWS::AccountId}:identity/
${identityName}",
          {
            "identityName": {
              "Ref": "IdentityName"
            }
          }
        ]
      },
      {
        "Fn::Sub": [
          "arn:${AWS::Partition}:ses:${AWS::Region}:${AWS::AccountId}:template/
${templateName}",
          {
            "templateName": {
              "Ref": "TemplateName"
            }
          }
        ]
      }
    ]
  },
  {
```

```
    "Action": [
      "ses:GetIdentityVerificationAttributes",
      "ses:VerifyEmailIdentity"
    ],
    "Effect": "Allow",
    "Resource": "*"
  }
]
```

SESCrudPolicy

授予发送电子邮件和验证身份的权限。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "ses:GetIdentityVerificationAttributes",
      "ses:SendEmail",
      "ses:SendRawEmail",
      "ses:VerifyEmailIdentity"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:ses:${AWS::Region}:${AWS::AccountId}:identity/
${identityName}",
        {
          "identityName": {
            "Ref": "IdentityName"
          }
        }
      ]
    }
  }
]
```

SESEmailTemplateCrudPolicy

授予创建、获取、列出、更新和删除 Amazon SES 电子邮件模板的权限。

```
"Statement": [
  {
    "Effect": "Allow",
```

```
    "Action": [
      "ses:CreateTemplate",
      "ses:GetTemplate",
      "ses:ListTemplates",
      "ses:UpdateTemplate",
      "ses:DeleteTemplate",
      "ses:TestRenderTemplate"
    ],
    "Resource": "*"
  }
]
```

SESSendBouncePolicy

SendBounce 授予亚马逊简单电子邮件服务 (Amazon SES) Service 身份的权限。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "ses:SendBounce"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:ses:${AWS::Region}:${AWS::AccountId}:identity/
${identityName}",
        {
          "identityName": {
            "Ref": "IdentityName"
          }
        }
      ]
    }
  }
]
```

SNSCrudPolicy

授予创建、发布和订阅 Amazon SNS 主题的权限。

```
"Statement": [
  {
```

```
    "Effect": "Allow",
    "Action": [
      "sns:ListSubscriptionsByTopic",
      "sns:CreateTopic",
      "sns:SetTopicAttributes",
      "sns:Subscribe",
      "sns:Publish"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:sns:${AWS::Region}:${AWS::AccountId}:${topicName}*",
        {
          "topicName": {
            "Ref": "TopicName"
          }
        }
      ]
    }
  }
]
```

SNSPublishMessagePolicy

授予将消息发布到 Amazon Simple Notification Service (Amazon SNS)主题的权限。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "sns:Publish"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:sns:${AWS::Region}:${AWS::AccountId}:${topicName}",
        {
          "topicName": {
            "Ref": "TopicName"
          }
        }
      ]
    }
  }
]
```

SQSPollerPolicy

授予轮询 Amazon Simple Queue Service (Amazon SQS) 队列的权限。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "sqs:ChangeMessageVisibility",
      "sqs:ChangeMessageVisibilityBatch",
      "sqs:DeleteMessage",
      "sqs:DeleteMessageBatch",
      "sqs:GetQueueAttributes",
      "sqs:ReceiveMessage"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:sqs:${AWS::Region}:${AWS::AccountId}:${queueName}",
        {
          "queueName": {
            "Ref": "QueueName"
          }
        }
      ]
    }
  }
]
```

SQSSendMessagePolicy

授予向 Amazon SQS 队列发送消息的权限。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "sqs:SendMessage*"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:sqs:${AWS::Region}:${AWS::AccountId}:${queueName}",
        {
          "queueName": {
```

```
            "Ref": "QueueName"
          }
        }
      ]
    }
  }
]
```

SSMParameterReadPolicy

授予访问来自 Amazon S EC2 ystems Manager (SSM) 参数存储库的参数的权限，以便在此账户中加载密钥。在参数名称不包含斜杠前缀时使用。

> **Note**
>
> 如果没有使用默认密钥，则还需要 `KMSDecryptPolicy` 策略。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "ssm:DescribeParameters"
    ],
    "Resource": "*"
  },
  {
    "Effect": "Allow",
    "Action": [
      "ssm:GetParameters",
      "ssm:GetParameter",
      "ssm:GetParametersByPath"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:ssm:${AWS::Region}:${AWS::AccountId}:parameter/
${parameterName}",
        {
          "parameterName": {
            "Ref": "ParameterName"
          }
        }
```

```
      ]
    }
  }
]
```

SSMParameterWithSlashPrefixReadPolicy

授予访问来自 Amazon S EC2 ystems Manager (SSM) 参数存储库的参数的权限，以便在此账户中加载密钥。在参数名称包含斜杠前缀时使用。

> **Note**
>
> 如果没有使用默认密钥，则还需要 `KMSDecryptPolicy` 策略。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "ssm:DescribeParameters"
    ],
    "Resource": "*"
  },
  {
    "Effect": "Allow",
    "Action": [
      "ssm:GetParameters",
      "ssm:GetParameter",
      "ssm:GetParametersByPath"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:ssm:${AWS::Region}:${AWS::AccountId}:parameter
${parameterName}",
        {
          "parameterName": {
            "Ref": "ParameterName"
          }
        }
      ]
    }
  }
```

```
]
```

StepFunctionsExecutionPolicy

授予开始执行 Step Functions 状态机的权限。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "states:StartExecution"
    ],
    "Resource": {
      "Fn::Sub": [
        "arn:${AWS::Partition}:states:${AWS::Region}:${AWS::AccountId}:stateMachine:
${stateMachineName}",
        {
          "stateMachineName": {
            "Ref": "StateMachineName"
          }
        }
      ]
    }
  }
]
```

TextractDetectAnalyzePolicy

授予使用 Amazon Textract 检测和分析文档的权限。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "textract:DetectDocumentText",
      "textract:StartDocumentTextDetection",
      "textract:StartDocumentAnalysis",
      "textract:AnalyzeDocument"
    ],
    "Resource": "*"
  }
]
```

TextractGetResultPolicy

授予从 Amazon Textract 中获取检测到和分析过的文档的权限。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "textract:GetDocumentTextDetection",
      "textract:GetDocumentAnalysis"
    ],
    "Resource": "*"
  }
]
```

TextractPolicy

授予对 Amazon Textract 的完全访问权限。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "textract:*"
    ],
    "Resource": "*"
  }
]
```

VPCAccessPolicy

授予访问权限以创建、删除、描述和分离弹性网络接口。

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "ec2:CreateNetworkInterface",
      "ec2:DeleteNetworkInterface",
      "ec2:DescribeNetworkInterfaces",
      "ec2:DetachNetworkInterface"
    ],
    "Resource": "*"
```

```
  }
]
```

使用 Amazon CloudFormation 机制管理 Amazon SAM 权限

要控制对 Amazon 资源的访问权限， Amazon Serverless Application Model (Amazon SAM) 可以使用与相同的机制 Amazon CloudFormation。有关更多信息，请参阅《Amazon CloudFormation 用户指南》中的使用 Amazon Identity and Access Management进行访问控制。

有三个选项可用于向用户授予管理无服务器应用程序的权限。每个选项可为用户提供不同级别的访问控制。

- 授予管理员权限。
- 附加必要的 Amazon 托管策略。
- 授予特定 Amazon Identity and Access Management (IAM) 权限。

根据您选择的选项，用户只能管理包含他们有权访问的 Amazon 资源的无服务器应用程序。

以下部分详细介绍了每个选项。

授予管理员权限

如果您向用户授予管理员权限，则他们可以管理包含任意 Amazon 资源组合的无服务器应用程序。这是最简单的选项，但也会向用户授予最广泛的权限，使得他们能够执行具有巨大影响的操作。

有关向用户授予管理员权限的详细信息，请参阅 IAM 用户指南中的创建您的第一个 IAM 管理员用户和组。

附加必要的 Amazon 托管策略

您可以使用 Amazon 托管策略向用户授予部分权限，而不是授予完全管理员权限。如果使用此选项，请确保 Amazon 托管策略集涵盖用户管理的无服务器应用程序所需的所有操作和资源。

例如，以下 Amazon 托管策略足以部署示例 Hello World 应用程序：

- AWSCloudFormationFullAccess
- IAMFull访问权限
- AWSLambda_FullAccess
- 亚马逊APIGateway管理员
- 亚马逊 3 FullAccess

- Amazon EC2 ContainerRegistryFullAccess

有关附加策略到 IAM 用户的信息，请参阅《IAM 用户指南》中的更改 IAM 用户的权限。

授予特定的 IAM 权限

要实现最精细级别的访问控制，您可以使用策略声明向用户授予特定的 IAM 权限。如果使用此选项，请确保策略声明包含用户管理的无服务器应用程序所需的所有操作和资源。

使用此选项的最佳做法是，拒绝用户创建角色（包括 Lambda 执行角色）的权限，这样他们就无法向自己授予升级权限。因此，作为管理员，您必须先创建 Lambda 执行角色（该角色将在用户要管理的无服务器应用程序中指定）。有关创建 Lambda 执行角色的信息，请参阅在 IAM 控制台中创建执行角色。

对于示例 Hello World 应用程序，足以运行该应用程序。AWSLambdaBasicExecutionRole创建 Lambda 执行角色后，修改示例 Hello World 应用程序的 Amazon SAM 模板文件，将以下属性添加到资源中`AWS::Serverless::Function`：

```
  Role: lambda-execution-role-arn
```

修改后的 Hello World 应用程序准备就绪后，以下策略声明会向用户授予部署、更新和删除应用程序所需的足够权限：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CloudFormationTemplate",
            "Effect": "Allow",
            "Action": [
                "cloudformation:CreateChangeSet"
            ],
            "Resource": [
                "arn:aws:cloudformation:region:aws:transform/Serverless-2016-10-31"
            ]
        },
        {
            "Sid": "CloudFormationStack",
            "Effect": "Allow",
            "Action": [
                "cloudformation:CreateChangeSet",
                "cloudformation:CreateStack",
```

```
                "cloudformation:DeleteStack",
                "cloudformation:DescribeChangeSet",
                "cloudformation:DescribeStackEvents",
                "cloudformation:DescribeStacks",
                "cloudformation:ExecuteChangeSet",
                "cloudformation:GetTemplateSummary",
                "cloudformation:ListStackResources",
                "cloudformation:UpdateStack"
            ],
            "Resource": [
                "arn:aws:cloudformation:region:111122223333:stack/stack-name"
            ]
        },
        {
            "Sid": "S3",
            "Effect": "Allow",
            "Action": [
                "s3:CreateBucket",
                "s3:GetObject",
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::name-of-bucket/key-name"
            ]
        },
        {
            "Sid": "ECRRepository",
            "Effect": "Allow",
            "Action": [
                "ecr:BatchCheckLayerAvailability",
                "ecr:BatchGetImage",
                "ecr:CompleteLayerUpload",
                "ecr:CreateRepository",
                "ecr:DeleteRepository",
                "ecr:DescribeImages",
                "ecr:DescribeRepositories",
                "ecr:GetDownloadUrlForLayer",
                "ecr:GetRepositoryPolicy",
                "ecr:InitiateLayerUpload",
                "ecr:ListImages",
                "ecr:PutImage",
                "ecr:SetRepositoryPolicy",
                "ecr:UploadLayerPart"
            ],
```

```
            "Resource": [
                "arn:aws:ecr:region:111122223333:repository/repository-name"
            ]
        },
        {
            "Sid": "ECRAuthToken",
            "Effect": "Allow",
            "Action": [
                "ecr:GetAuthorizationToken"
            ],
            "Resource": [
                "arn:aws:ecr:region:111122223333:repository/repository-name"
            ]
        },
        {
            "Sid": "Lambda",
            "Effect": "Allow",
            "Action": [
                "lambda:AddPermission",
                "lambda:CreateFunction",
                "lambda:DeleteFunction",
                "lambda:GetFunction",
                "lambda:GetFunctionConfiguration",
                "lambda:ListTags",
                "lambda:RemovePermission",
                "lambda:TagResource",
                "lambda:UntagResource",
                "lambda:UpdateFunctionCode",
                "lambda:UpdateFunctionConfiguration"
            ],
            "Resource": [
                "arn:aws:lambda:region:111122223333:function/function-name"
            ]
        },
        {
            "Sid": "IAM",
            "Effect": "Allow",
            "Action": [
                "iam:CreateRole",
                "iam:AttachRolePolicy",
                "iam:DeleteRole",
                "iam:DetachRolePolicy",
                "iam:GetRole",
                "iam:TagRole"
```

```
            ],
            "Resource": [
                "arn:aws:iam::111122223333:role/role-name"
            ]
        },
        {
            "Sid": "IAMPassRole",
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "arn:aws:iam::111122223333:role/role-name",
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": "lambda.amazonaws.com"
                }
            }
        },
        {
            "Sid": "APIGateway",
            "Effect": "Allow",
            "Action": [
                "apigateway:DELETE",
                "apigateway:GET",
                "apigateway:PATCH",
                "apigateway:POST",
                "apigateway:PUT"
            ],
            "Resource": [
                "arn:aws:apigateway:region:api-id:resource-path"
            ]
        }
    ]
}
```

> **Note**
>
> 本节中的示例策略声明向您授予部署、更新和删除示例 Hello World 应用程序所需的足够权限。如果您向应用程序添加其他资源类型，则需要更新政策声明以包含以下内容：
>
> 1. 应用程序调用服务操作所需的权限。
> 2. 服务主体（如果对于服务操作是必需的）。

例如，如果您添加 Step Functions 工作流程，则可能需要添加此处所列操作的权限以及 `states.amazonaws.com` 服务主体。

有关 IAM policy 的更多信息，请参阅 IAM 用户指南中的管理 IAM policy。

使用您的 Amazon SAM 模板控制 API 访问权限

控制 API Gateway 的访问权限 APIs 有助于确保您的无服务器应用程序是安全的，并且只能通过您启用的授权进行访问。您可以在 Amazon SAM 模板中启用授权，以控制谁可以访问您的 API Gateway APIs。

Amazon SAM 支持多种控制您的 API Gateway 访问权限的机制 APIs。`AWS::Serverless::HttpApi` 和 `AWS::Serverless::Api` 资源类型所支持的机制集有所不同。

下表总结了每种资源类型支持的机制。

控制访问的机制	AWS::Serverless::HttpApi	AWS::Serverless::Api
Lambda 授权方	✓	✓
IAM 权限		✓
Amazon Cognito 用户群体	✓ *	✓
API 密钥		✓
资源策略		✓
OAuth 2.0/JWT 授权方	✓	

* 您可以使用 Amazon Cognito 作为 `AWS::Serverless::HttpApi` 资源类型的 JSON Web Token (JWT) 颁发者。

- Lambda 授权方 – Lambda 授权方（以前称为自定义授权方）是您为控制对 API 的访问而提供的 Lambda 函数。调用 API 时，将使用客户端应用程序提供的请求上下文或授权令牌调用此 Lambda 函数。Lambda 函数会响应调用者是否有权执行所请求的操作。

`AWS::Serverless::HttpApi` 和 `AWS::Serverless::Api` 资源类型均支持 Lambda 授权方。

有关使用的 Lambda 授权方的更多信息*`AWS::Serverless::HttpApi`*，请参阅 APIs API Gateway 开发者指南中的使用 HTTP 的 Amazon Lambda 授权方。有关 Lambda 授权方与 `AWS::Serverless::Api` 的更多信息，请参阅《API Gateway 开发人员指南》中的使用 API Gateway Lambda 授权方。

有关任一资源类型的 Lambda 授权方的示例，请参阅 Lambda 授权方示例 Amazon SAM。

- IAM 权限 – 您可以使用 Amazon Identity and Access Management (IAM) 权限控制谁可以调用您的 API。调用 API 的用户必须使用 IAM 凭证进行身份验证。只有存在与代表 API 调用方的 IAM 用户、包含该用户的 IAM 群组或该用户担任的 IAM 角色关联的 IAM policy 时，对 API 的调用才会成功。

  仅 `AWS::Serverless::Api` 资源类型支持 IAM 权限。

  有关更多信息，请参阅《API Gateway 开发人员指南》中的使用 IAM 权限控制对 API 的访问。有关示例，请参阅的 IAM 权限示例 Amazon SAM。
- Amazon Cognito 用户群体 – Amazon Cognito 用户群体是 Amazon Cognito 中的用户目录。API 的客户端必须先将用户登录到用户群体，然后为该用户获取身份或访问令牌。然后，客户端使用其中返回的令牌调用 API。只有在所需的令牌有效时，API 调用才会成功。

  `AWS::Serverless::Api` 资源类型支持 Amazon Cognito 用户群体。`AWS::Serverless::HttpApi` 资源类型支持使用 Amazon Cognito 作为 JWT 颁发者。

  有关更多信息，请参阅《API Gateway 开发人员指南》中的使用 Amazon Cognito 用户群体作为授权方控制对 REST API 的访问。有关示例，请参阅Amazon Cognito 用户池示例 Amazon SAM。
- API 密钥 – API 密钥是字母数字字符串值，可将它分发给应用程序开发人员（要向其授予对您的 API 的访问权的客户）。

  仅 `AWS::Serverless::Api` 资源类型支持 API 密钥。

  有关 API 密钥的更多信息，请参阅《API Gateway 开发人员指南》中的创建和使用带 API 密钥的使用计划。有关 API 密钥的示例，请参阅 的 API 密钥示例 Amazon SAM。
- 资源策略 – 资源策略是您可以附加到 API Gateway API 的 JSON 策略文档。使用资源策略控制指定的主体（通常是 IAM 用户或角色）能否调用 API。

  只有`AWS::Serverless::Api`资源类型支持资源策略作为控制 API Gateway 访问权限的机制 APIs。

有关资源策略的更多信息，请参阅《API Gateway 开发人员指南》中的使用 API Gateway 资源策略控制 API 的访问权限。有关资源策略的示例，请参阅 的资源策略示例 Amazon SAM。

- OAuth 2.0/JWT 授权者 — 你可以作 JWTs 为 OpenID C onnect (OIDC) 和 OAuth 2.0 框架的一部分来控制对你的访问权限。 APIsAPI Gateway JWTs 会验证客户端通过 API 请求提交的内容，并根据令牌验证和令牌中的范围（可选）允许或拒绝请求。

  只有`AWS::Serverless::HttpApi`资源类型支持 OAuth 2.0/JWT 授权者。

  有关更多信息，请参阅《API Gateway 开发者指南》中的 “ APIs 使用 JWT 授权者控制 HTTP 访问权限”。有关示例，请参阅OAuth 2.0/JWT 授权方示例 Amazon SAM。

选择控制访问的机制

您选择用于控制 API Gateway 访问权限的机制 APIs 取决于几个因素。例如，如果您有一个未设置授权或访问控制的全新项目，那么 Amazon Cognito 用户群体可能是您的最佳选择。这是因为在设置用户群体时，还会自动设置身份验证和访问控制。

但是，如果应用程序已经设置了身份验证，那么使用 Lambda 授权方可能是您的最佳选择。这是因为您可以调用现有的身份验证服务并根据响应返回策略文档。此外，如果您的应用程序需要用户群体不支持的自定义身份验证或访问控制逻辑，那么 Lambda 授权方可能是您的最佳选择。

选择使用哪种机制后，请参阅中的示例相应部分，了解如何使用该机制 Amazon SAM 来配置您的应用程序。

自定义错误响应

您可以使用 Amazon SAM 自定义某些 API Gateway 错误响应的内容。仅 `AWS::Serverless::Api` 资源类型支持自定义 API Gateway 响应。

有关 API Gateway 响应的更多信息，请参阅《API Gateway 开发人员指南》中的 API Gateway 中的网关响应。有关自定义响应的示例，请参阅 的自定义响应示例 Amazon SAM。

示例

- Lambda 授权方示例 Amazon SAM
- 的 IAM 权限示例 Amazon SAM
- Amazon Cognito 用户池示例 Amazon SAM
- 的 API 密钥示例 Amazon SAM

- 的资源策略示例 Amazon SAM
- OAuth 2.0/JWT 授权方示例 Amazon SAM
- 的自定义响应示例 Amazon SAM

Lambda 授权方示例 Amazon SAM

`AWS::Serverless::Api` 资源类型支持两种类型的 Lambda 授权方：`TOKEN` 授权方和 `REQUEST` 授权方。`AWS::Serverless::HttpApi` 资源类型仅支持 `REQUEST` 授权方。下面是每个类型的示例。

Lambda **`TOKEN`** 授权方示例 (AWS::Serverless::Api)

您可以 APIs 通过在模板中定义 Lambda `TOKEN` 授权机构来控制对您的访问权限。 Amazon SAM 为此，您需要使用 ApiAuth 数据类型。

以下是 Lambda `TOKEN` 授权方的示例 Amazon SAM 模板部分：

> **Note**
>
> 在以下示例中，SAM `FunctionRole` 是隐式生成的。

```
Resources:
  MyApi:
    Type: AWS::Serverless::Api
    Properties:
      StageName: Prod
      Auth:
        DefaultAuthorizer: MyLambdaTokenAuthorizer
        Authorizers:
          MyLambdaTokenAuthorizer:
            FunctionArn: !GetAtt MyAuthFunction.Arn

  MyFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: ./src
      Handler: index.handler
      Runtime: nodejs12.x
      Events:
        GetRoot:
```

```
          Type: Api
          Properties:
            RestApiId: !Ref MyApi
            Path: /
            Method: get

  MyAuthFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: ./src
      Handler: authorizer.handler
      Runtime: nodejs12.x
```

有关 Lambda 授权方的更多信息，请参阅《API Gateway 开发人员指南》中的使用 API Gateway Lambda 授权方。

Lambda **REQUEST** 授权方示例 (AWS::Serverless::Api)

您可以 APIs 通过在模板中定义 Lambda REQUEST 授权机构来控制对您的访问权限。 Amazon SAM 为此，您需要使用 ApiAuth 数据类型。

以下是 Lambda REQUEST 授权方的示例 Amazon SAM 模板部分：

```
Resources:
  MyApi:
    Type: AWS::Serverless::Api
    Properties:
      StageName: Prod
      Auth:
        DefaultAuthorizer: MyLambdaRequestAuthorizer
        Authorizers:
          MyLambdaRequestAuthorizer:
            FunctionPayloadType: REQUEST
            FunctionArn: !GetAtt MyAuthFunction.Arn
            Identity:
              QueryStrings:
                - auth

  MyFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: ./src
      Handler: index.handler
```

```
      Runtime: nodejs12.x
      Events:
        GetRoot:
          Type: Api
          Properties:
            RestApiId: !Ref MyApi
            Path: /
            Method: get

  MyAuthFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: ./src
      Handler: authorizer.handler
      Runtime: nodejs12.x
```

有关 Lambda 授权方的更多信息，请参阅《API Gateway 开发人员指南》中的使用 API Gateway Lambda 授权方。

Lambda 授权方示例 (AWS::Serverless::HttpApi)

您可以 APIs 通过在模板中定义 Lambda 授权机构来控制对 HTTP 的访问。 Amazon SAM 为此，您需要使用 HttpApiAuth 数据类型。

以下是 Lambda 授权方的示例 Amazon SAM 模板部分：

```
Resources:
  MyApi:
    Type: AWS::Serverless::HttpApi
    Properties:
      StageName: Prod
      Auth:
        DefaultAuthorizer: MyLambdaRequestAuthorizer
        Authorizers:
          MyLambdaRequestAuthorizer:
            FunctionArn: !GetAtt MyAuthFunction.Arn
            FunctionInvokeRole: !GetAtt MyAuthFunctionRole.Arn
            Identity:
              Headers:
                - Authorization
            AuthorizerPayloadFormatVersion: 2.0
            EnableSimpleResponses: true
```

```
  MyFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: ./src
      Handler: index.handler
      Runtime: nodejs12.x
      Events:
        GetRoot:
          Type: HttpApi
          Properties:
            ApiId: !Ref MyApi
            Path: /
            Method: get
            PayloadFormatVersion: "2.0"

  MyAuthFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: ./src
      Handler: authorizer.handler
      Runtime: nodejs12.x
```

的 IAM 权限示例 Amazon SAM

您可以 APIs 通过在 Amazon SAM 模板中定义 IAM 权限来控制对您的访问权限。为此，您需要使用 ApiAuth 数据类型。

以下是用于 IAM 权限的示例 Amazon SAM 模板：

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Resources:
  MyApi:
    Type: AWS::Serverless::Api
    Properties:
      StageName: Prod
      Description: 'API with IAM authorization'
      Auth:
        DefaultAuthorizer: AWS_IAM #sets AWS_IAM auth for all methods in this API
  MyFunction:
    Type: AWS::Serverless::Function
    Properties:
      Handler: index.handler
```

```
      Runtime: python3.10
      Events:
        GetRoot:
          Type: Api
          Properties:
            RestApiId: !Ref MyApi
            Path: /
            Method: get
      InlineCode: |
        def handler(event, context):
          return {'body': 'Hello World!', 'statusCode': 200}
```

有关 IAM 权限的更多信息，请参阅《API Gateway 开发人员指南》中的控制调用 API 的访问权限。

Amazon Cognito 用户池示例 Amazon SAM

您可以 APIs 通过在模板中定义 Amazon Cognito 用户池来控制对您的 Amazon SAM 访问权限。为此，您需要使用 ApiAuth 数据类型。

以下是用户池的示例 Amazon SAM 模板部分：

```
Resources:
  MyApi:
    Type: AWS::Serverless::Api
    Properties:
      StageName: Prod
      Cors: "'*'"
      Auth:
        DefaultAuthorizer: MyCognitoAuthorizer
        Authorizers:
          MyCognitoAuthorizer:
            UserPoolArn: !GetAtt MyCognitoUserPool.Arn

  MyFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: ./src
      Handler: lambda.handler
      Runtime: nodejs12.x
      Events:
        Root:
          Type: Api
          Properties:
```

```
            RestApiId: !Ref MyApi
            Path: /
            Method: GET

  MyCognitoUserPool:
    Type: AWS::Cognito::UserPool
    Properties:
      UserPoolName: !Ref CognitoUserPoolName
      Policies:
        PasswordPolicy:
          MinimumLength: 8
      UsernameAttributes:
        - email
      Schema:
        - AttributeDataType: String
          Name: email
          Required: false

  MyCognitoUserPoolClient:
    Type: AWS::Cognito::UserPoolClient
    Properties:
      UserPoolId: !Ref MyCognitoUserPool
      ClientName: !Ref CognitoUserPoolClientName
      GenerateSecret: false
```

有关 Amazon Cognito 用户群体的更多信息，请参阅《API Gateway 开发人员指南》中的使用 Amazon Cognito 用户群体作为授权方控制对 REST API 的访问。

的 API 密钥示例 Amazon SAM

您可以 APIs 通过在 Amazon SAM 模板中要求 API 密钥来控制对您的访问权限。为此，您需要使用 ApiAuth 数据类型。

以下是 API 密钥的示例 Amazon SAM 模板部分：

```
Resources:
  MyApi:
    Type: AWS::Serverless::Api
    Properties:
      StageName: Prod
      Auth:
        ApiKeyRequired: true # sets for all methods
```

```
  MyFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: .
      Handler: index.handler
      Runtime: nodejs12.x
      Events:
        ApiKey:
          Type: Api
          Properties:
            RestApiId: !Ref MyApi
            Path: /
            Method: get
            Auth:
              ApiKeyRequired: true
```

有关 API 密钥的更多信息，请参阅《API Gateway 开发人员指南》中的创建和使用带 API 密钥的使用计划。

的资源策略示例 Amazon SAM

您可以 APIs 通过在 Amazon SAM 模板中附加资源策略来控制对您的访问权限。为此，您需要使用 ApiAuth 数据类型。

以下是私有 API 的示例 Amazon SAM 模板。私有 API 必须具有资源策略才能部署。

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Resources:
  MyPrivateApi:
    Type: AWS::Serverless::Api
    Properties:
      StageName: Prod
      EndpointConfiguration: PRIVATE  # Creates a private API. Resource policies are
 required for all private APIs.
      Auth:
        ResourcePolicy:
          CustomStatements:
            - Effect: 'Allow'
              Action: 'execute-api:Invoke'
              Resource: ['execute-api:/*/*/*']
              Principal: '*'
            - Effect: 'Deny'
```

```
              Action: 'execute-api:Invoke'
              Resource: ['execute-api:/*/*/*']
              Principal: '*'
  MyFunction:
    Type: 'AWS::Serverless::Function'
    Properties:
      InlineCode: |
        def handler(event, context):
          return {'body': 'Hello World!', 'statusCode': 200}
      Handler: index.handler
      Runtime: python3.10
      Events:
        AddItem:
          Type: Api
          Properties:
            RestApiId:
              Ref: MyPrivateApi
            Path: /
            Method: get
```

有关资源策略的更多信息，请参阅《API Gateway 开发人员指南》中的使用 API Gateway 资源策略控制 API 的访问权限。有关私有化的更多信息 APIs，请参阅 API Gateway 开发者指南中的在 Amazon API Gateway 中创建私有 API。

OAuth 2.0/JWT 授权方示例 Amazon SAM

JWTs 作为 OpenID Connect (OIDC) 和 2.0 框架的一部分，你可以控制对你的 APIs 使用的访问权限。OAuth 为此，您需要使用 HttpApiAuth 数据类型。

以下是 OAuth 2.0/JWT 授权方的示例 Amazon SAM 模板部分：

```
Resources:
  MyApi:
    Type: AWS::Serverless::HttpApi
    Properties:
      Auth:
        Authorizers:
          MyOauth2Authorizer:
            AuthorizationScopes:
              - scope
            IdentitySource: $request.header.Authorization
            JwtConfiguration:
              audience:
```

```
                - audience1
                - audience2
              issuer: "https://www.example.com/v1/connect/oidc"
        DefaultAuthorizer: MyOauth2Authorizer
      StageName: Prod
  MyFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: ./src
      Events:
        GetRoot:
          Properties:
            ApiId: MyApi
            Method: get
            Path: /
            PayloadFormatVersion: "2.0"
          Type: HttpApi
      Handler: index.handler
      Runtime: nodejs12.x
```

有关 OAuth 2.0/JWT 授权方的更多信息，请参阅《API Gateway 开发者指南》中的 “ APIs 使用 JWT 授权者控制 HTTP 访问权限”。

的自定义响应示例 Amazon SAM

您可以通过在 Amazon SAM 模板中定义响应标头来自定义某些 API Gateway 错误响应。为此，您需要使用网关响应对象数据类型。

以下是为`DEFAULT_5XX`错误创建自定义响应的示例 Amazon SAM 模板。

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Resources:
  MyApi:
    Type: AWS::Serverless::Api
    Properties:
      StageName: Prod
      GatewayResponses:
        DEFAULT_5XX:
          ResponseParameters:
            Headers:
              Access-Control-Expose-Headers: "'WWW-Authenticate'"
              Access-Control-Allow-Origin: "'*'"
```

```
              ErrorHeader: "'MyCustomErrorHeader'"
          ResponseTemplates:
            application/json: "{\"message\": \"Error on the $context.resourcePath
resource\" }"

 GetFunction:
   Type: AWS::Serverless::Function
   Properties:
     Runtime: python3.10
     Handler: index.handler
     InlineCode: |
       def handler(event, context):
         raise Exception('Check out the new response!')
     Events:
       GetResource:
         Type: Api
         Properties:
           Path: /error
           Method: get
           RestApiId: !Ref MyApi
```

有关 API Gateway 响应的更多信息，请参阅《API Gateway 开发人员指南》中的 API Gateway 中的网关响应。

使用 Lambda 图层提高效率 Amazon SAM

使用 Amazon SAM，您可以在无服务器应用程序中包含图层。 Amazon Lambda 层允许您将代码从 Lambda 函数提取到 Lambda 层中，然后可以在多个 Lambda 函数中使用该层。这样做可以让您缩小部署包的大小，将核心函数逻辑与依赖项分开，并在多个函数之间共享依赖项。有关层的更多信息，请参阅《Amazon Lambda 开发人员指南》中的 Lambda 层。

本主题提供有关以下内容的信息：

- 在应用程序中包含层
- 如何在本地缓存层

有关构建自定义层的更多信息，请参阅 在中构建 Lambda 图层 Amazon SAM。

在应用程序中包含层

要在应用程序中包含层，请使用 AWS::Serverless::Function 资源类型的 `Layers` 属性。

以下是带有包含层的 Lambda 函数的示例 Amazon SAM 模板：

```
ServerlessFunction:
  Type: AWS::Serverless::Function
  Properties:
    CodeUri: .
    Handler: my_handler
    Runtime: Python3.7
    Layers:
        - <LayerVersion ARN>
```

如何在本地缓存层

当您使用其中一个 `sam local` 命令调用函数时，函数的层包会被下载并缓存在本地主机上。

下表显示不同操作系统的默认缓存目录位置。

OS	位置
Windows 7	`C:\Users\<user>\AppData\Roaming\AWS SAM`
Windows 8	`C:\Users\<user>\AppData\Roaming\AWS SAM`
Windows 10	`C:\Users\<user>\AppData\Roaming\AWS SAM`
macOS	`~/.aws-sam/layers-pkg`
Unix	`~/.aws-sam/layers-pkg`

缓存软件包后， Amazon SAM CLI 将图层叠加到用于调用函数的 Docker 镜像上。的 Amazon SAM CLI 生成其构建的图像以及缓存中保存 LayerVersions 的图像的名称。您可以在以下部分找到有关架构的更多详细信息。

要检查叠加层，请执行以下命令在要检查的映像中启动 bash 会话：

```
docker run -it --entrypoint=/bin/bash samcli/lambda:<Tag following the schema outlined
 in Docker Image Tag Schema> -i
```

层缓存目录名称架构

给 LayerVersionArn 定模板中定义的， Amazon SAM CLI 从 AR LayerName N 中提取和版本。它会创建一个目录，用于存放名为 `LayerName-Version-<first 10 characters of sha256 of ARN>` 的层内容。

示例：

```
ARN = arn:aws:lambda:us-west-2:111111111111:layer:myLayer:1
Directory name = myLayer-1-926eeb5ff1
```

Docker 映像标签架构

要计算唯一图层哈希，请将所有唯一的图层名称与分隔符 “-” 合并，取 SHA256 哈希，然后取前 10 个字符。

示例：

```
ServerlessFunction:
  Type: AWS::Serverless::Function
  Properties:
    CodeUri: .
    Handler: my_handler
    Runtime: Python3.7
    Layers:
        - arn:aws:lambda:us-west-2:111111111111:layer:myLayer:1
        - arn:aws:lambda:us-west-2:111111111111:layer:mySecondLayer:1
```

唯一名称的计算方法与层缓存目录名称架构相同：

```
arn:aws:lambda:us-west-2:111111111111:layer:myLayer:1 = myLayer-1-926eeb5ff1
arn:aws:lambda:us-west-2:111111111111:layer:mySecondLayer:1 =
 mySecondLayer-1-6bc1022bdf
```

要计算唯一层哈希，请将所有唯一的层名称合并，分隔符为 '-'，取 sha256 哈希，然后取前 25 个字符：

```
myLayer-1-926eeb5ff1-mySecondLayer-1-6bc1022bdf = 2dd7ac5ffb30d515926aef
```

然后将此值与函数的运行时和架构合并，分隔符为 '-'：

```
python3.7-x86_64-2dd7ac5ffb30d515926aefffd
```

使用嵌套应用程序重用代码和资源 Amazon SAM

无服务器应用程序可以包含一个或多个嵌套应用程序。嵌套应用程序是大型应用程序的一部分，可以作为独立构件或大型应用程序的组件进行打包和部署。嵌套应用程序允许您将常用代码转换为其自己的应用程序，然后可以在大型无服务器应用程序或多个无服务器应用程序中重复使用。

随着无服务器架构的发展，通常会出现常见的模式，其中在多个应用程序模板中定义相同组件。嵌套应用程序允许您在单独的 Amazon SAM 模板中重复使用常用代码、功能、资源和配置，从而只能维护来自单一来源的代码。这可以减少重复的代码和配置。此外，这种模块化方法简化了开发过程，增强了代码组织，并促进了无服务器应用程序之间的一致性。使用嵌套应用程序，您可以更加专注于应用程序特有的业务逻辑。

要在无服务器应用程序中定义嵌套应用程序，请使用 AWS::Serverless::Application 资源类型。

您可以从以下两个来源定义嵌套应用程序：

- Amazon Serverless Application Repository 应用程序 – 您可以使用您的账户在 Amazon Serverless Application Repository中可用的应用程序来定义嵌套应用程序。它们可以是您账户中的私有应用程序、与您的账户私下共享的应用程序或在 Amazon Serverless Application Repository中公开共享的应用程序。有关不同部署权限级别的更多信息，请参阅《Amazon Serverless Application Repository 开发人员指南》中的应用程序部署权限和发布应用程序。
- 本地应用程序 – 您可以使用存储在本地文件系统上的应用程序来定义嵌套应用程序。

有关如何在无服务器应用程序中使用 Amazon SAM 定义这两种类型的嵌套应用程序的详细信息，请参阅以下各节。

> **Note**
>
> 无服务器应用程序中可以嵌套的最大应用程序数量为 200。
> 嵌套应用程序可以拥有的最大参数数量为 60。

从中定义嵌套应用程序 Amazon Serverless Application Repository

您可以使用 Amazon Serverless Application Repository中可用的应用程序来定义嵌套应用程序。您还可以使用 Amazon Serverless Application Repository存储和分发包含嵌套应用程序的应用程序。要查看中嵌套应用程序的详细信息 Amazon Serverless Application Repository，您可以使用 Amazon 软件开发工具包 Amazon CLI、或 Lambda 控制台。

要在无服务器应用程序的 Amazon SAM 模板中定义托管 Amazon Serverless Application Repository 在中的应用程序，请使用每个 Amazon Serverless Application Repository 应用程序详细信息页面上的 “复制为 SAM 资源” 按钮。为此，请按照以下步骤操作：

1. 请确保您已登录 Amazon Web Services Management Console。
2. 使用《Amazon Serverless Application Repository 开发人员指南》 Amazon Serverless Application Repository 的 “浏览、搜索和部署应用程序” 部分中的步骤，找到要嵌套的应用程序。
3. 选择复制为 SAM 资源按钮。您正在查看的应用程序的 SAM 模板部分现在位于剪贴板中。
4. 将 SAM 模板部分粘贴到要嵌套在此应用程序中的应用程序的 SAM 模板文件的 `Resources:` 部分。

以下是托管在 Amazon Serverless Application Repository中的嵌套应用程序的 SAM 模板部分示例：

```
Transform: AWS::Serverless-2016-10-31

Resources:
  applicationaliasname:
    Type: AWS::Serverless::Application
    Properties:
      Location:
        ApplicationId: arn:aws:serverlessrepo:us-
east-1:123456789012:applications/application-alias-name
        SemanticVersion: 1.0.0
      Parameters:
        # Optional parameter that can have default value overridden
        # ParameterName1: 15 # Uncomment to override default value
        # Required parameter that needs value to be provided
        ParameterName2: YOUR_VALUE
```

如果没有必需的参数设置，则可以省略模板的 `Parameters:` 部分。

> **Important**
>
> 包含托管在中的嵌套应用程序的应用程序会 Amazon Serverless Application Repository 继承嵌套应用程序的共享限制。
> 例如，假设一个应用程序是公开共享的，但它包含一个仅与创建父应用程序的 Amazon 账户私下共享的嵌套应用程序。在这种情况下，如果您的 Amazon 账户无权部署嵌套应用程序，

则无法部署父应用程序。有关应用程序部署权限的更多信息，请参阅《Amazon Serverless Application Repository 开发人员指南》中的应用程序部署权限和发布应用程序。

从本地文件系统定义嵌套应用程序

您可以使用存储在本地文件系统上的应用程序来定义嵌套应用程序。为此，您可以指定存储在本地文件系统中的 Amazon SAM 模板文件的路径。

以下是嵌套本地应用程序的 SAM 模板部分示例：

```
Transform: AWS::Serverless-2016-10-31

Resources:
  applicationaliasname:
    Type: AWS::Serverless::Application
    Properties:
      Location: ../my-other-app/template.yaml
      Parameters:
        # Optional parameter that can have default value overridden
        # ParameterName1: 15 # Uncomment to override default value
        # Required parameter that needs value to be provided
        ParameterName2: YOUR_VALUE
```

如果没有参数设置，则可以省略模板的 `Parameters:` 部分。

部署嵌套应用程序

您可以通过以下方式部署嵌套应用程序 Amazon SAM CLI 命令`sam deploy`。有关更多详细信息，请参阅 使用部署您的应用程序和资源 Amazon SAM。

> **Note**
>
> 部署包含嵌套应用程序的应用程序时，必须确认其包含嵌套应用程序。您可以通过传递`CAPABILITY_AUTO_EXPAND`给 CreateCloudFormationChangeSet API 或使用`aws serverlessrepo create-cloud-formation-change-set` Amazon CLI 命令来执行此操作。
>
> 有关确认嵌套应用程序的更多信息，请参阅《Amazon Serverless Application Repository 开发人员指南》中的在部署应用程序时确认 IAM 角色、资源策略和嵌套应用程序。

使用 “ EventBridge调度器” 管理基于时间的事件 Amazon SAM

本主题中的内容详细介绍了什么是 Amazon S EventBridge cheduler、 Amazon SAM 提供哪些支持、如何创建计划程序事件，以及在创建计划程序事件时可以参考的示例。

什么是 Amazon EventBridge 日程安排？

使用 EventBridge 日程安排器在 Amazon SAM 模板中安排活动。Amazon S EventBridge cheduler 是一项计划服务，允许您在所有 Amazon 服务中创建、启动和管理数千万个事件和任务。此服务对于与时间相关的事件特别有用。您可以使用它来计划事件和基于时间的重复调用。它还支持一次性事件以及带有开始和结束时间的 rate 和 chron 表达式。

要了解有关 Amazon EventBridge 日程安排器的更多信息，请参阅什么是亚马逊 EventBridge 日程安排？ 在《EventBridge 日程安排器用户指南》中。

主题

- EventBridge 中的调度器支持 Amazon SAM
- 在中创建 EventBridge 调度器事件 Amazon SAM
- 示例
- 了解更多

EventBridge 中的调度器支持 Amazon SAM

Amazon Serverless Application Model (Amazon SAM) 模板规范提供了一种简单的简短语法，您可以使用该语法通过 S EventBridge cheduler 为 Amazon Lambda 和安排事件。 Amazon Step Functions

在中创建 EventBridge 调度器事件 Amazon SAM

在 Amazon SAM 模板中将该`ScheduleV2`属性设置为事件类型，以定义您的 EventBridge 日程安排器事件。此属性支持 `AWS::Serverless::Function` 和 `AWS::Serverless::StateMachine` 资源类型。

```
MyFunction:
  Type: AWS::Serverless::Function
  Properties:
    Events:
      CWSchedule:
```

```
        Type: ScheduleV2
        Properties:
          ScheduleExpression: 'rate(1 minute)'
          Name: TestScheduleV2Function
          Description: Test schedule event

MyStateMachine:
  Type: AWS::Serverless::StateMachine
  Properties:
    Events:
      CWSchedule:
        Type: ScheduleV2
        Properties:
          ScheduleExpression: 'rate(1 minute)'
          Name: TestScheduleV2StateMachine
          Description: Test schedule event
```

EventBridge 调度器事件调度还支持未处理事件的死信队列 (DLQ)。有关死信队列的更多信息，请参阅《日程安排器用户指南》中的为日 EventBridge 程安排器配置死信队列。EventBridge

指定 DLQ ARN 后 Amazon SAM ，配置调度程序计划向 DLQ 发送消息的权限。如果未指定 DLQ ARN Amazon SAM ，则将创建 DLQ 资源。

示例

使用定义 EventBridge 调度器事件的基本示例 Amazon SAM

```
Transform: AWS::Serverless-2016-10-31
Resources:
  MyLambdaFunction:
    Type: AWS::Serverless::Function
    Properties:
      Handler: index.handler
      Runtime: python3.8
      InlineCode: |
        def handler(event, context):
            print(event)
            return {'body': 'Hello World!', 'statusCode': 200}
      MemorySize: 128
      Events:
        Schedule:
          Type: ScheduleV2
          Properties:
```

```
            ScheduleExpression: rate(1 minute)
            Input: '{"hello": "simple"}'

  MySFNFunction:
    Type: AWS::Serverless::Function
    Properties:
      Handler: index.handler
      Runtime: python3.8
      InlineCode: |
        def handler(event, context):
            print(event)
            return {'body': 'Hello World!', 'statusCode': 200}
      MemorySize: 128

  StateMachine:
    Type: AWS::Serverless::StateMachine
    Properties:
      Type: STANDARD
      Definition:
        StartAt: MyLambdaState
        States:
          MyLambdaState:
            Type: Task
            Resource: !GetAtt MySFNFunction.Arn
            End: true
      Policies:
        - LambdaInvokePolicy:
            FunctionName: !Ref MySFNFunction
      Events:
        Events:
        Schedule:
          Type: ScheduleV2
          Properties:
            ScheduleExpression: rate(1 minute)
            Input: '{"hello": "simple"}'
```

了解更多

要了解有关定义ScheduleV2 EventBridge 调度器属性的更多信息，请参阅：

- 用于 `AWS::Serverless::Function` 的 ScheduleV2。
- 用于 `AWS::Serverless::StateMachine` 的 ScheduleV2。

使用编排资源 Amazon SAMAmazon Step Functions

您可以使用Amazon Step Functions来编排 Amazon Lambda 函数和其他 Amazon 资源，以形成复杂而强大的工作流程。Step Functions 告诉您的应用程序何时以及在什么条件下使用您的 Amazon 资源（如 Amazon Lambda 函数）。这简化了形成复杂而稳健的工作流的过程。使用 AWS::Serverless::StateMachine，您可以定义工作流中的各个步骤，关联每个步骤中的资源，然后将这些步骤排列在一起。您还可以在需要的地方添加转换和条件。这简化了制定复杂而稳健的工作流的过程。

> **Note**
>
> 要管理包含 Step Functions 状态机的 Amazon SAM 模板，必须使用 0.52.0 或更高版本的 Amazon SAM CLI。 要检查你有哪个版本，请执行命令`sam --version`。

Step Functions 是基于任务和状态机的概念。您使用基于 JSON 的 Amazon States Language 定义状态机。Step Functions 控制台显示状态机结构的图形视图，因此您能够直观检查状态机逻辑和监控执行。

借助 Amazon Serverless Application Model (Amazon SAM) 中的 Step Functions 支持，你可以执行以下操作：

- 定义状态机，可以直接在 Amazon SAM 模板中定义，也可以在单独的文件中定义
- 通过 Amazon SAM 策略模板、内联策略或托管策略创建状态机执行角色
- 使用 API Gateway 或 Amazon EventBridge 事件、在 Amazon SAM 模板内按计划或 APIs 直接调用触发状态机执行
- 使用可用的 Amazon SAM 策略模板创建常见的 Step Functions 开发模式。

示例

以下 Amazon SAM 模板文件中的示例片段在定义文件中定义了 Step Functions 状态机。请注意，该 `my_state_machine.asl.json` 文件必须以 Amazon States Language 编写。

```
AWSTemplateFormatVersion: "2010-09-09"
Transform: AWS::Serverless-2016-10-31
Description: Sample SAM template with Step Functions State Machine
```

```
Resources:
  MyStateMachine:
    Type: AWS::Serverless::StateMachine
    Properties:
      DefinitionUri: statemachine/my_state_machine.asl.json
      ...
```

要下载包含 Step Functions 状态机的示例 Amazon SAM 应用程序，请参阅《Amazon Step Functions 开发者指南》 Amazon SAM中的 “使用创建 Step Functions 状态机”。

更多信息

要了解有关 Step Functions 及其与之配合使用的更多信息 Amazon SAM，请参阅以下内容：

- Amazon Step Functions 的工作原理
- Amazon Step Functions 和 Amazon Serverless Application Model
- 教程：使用创建 Step Functions 状态机 Amazon SAM
- Amazon SAM 规格： AWS::Serverless::StateMachine

为您的 Amazon SAM 应用程序设置代码签名

为确保仅部署可信代码，您可以使用 Amazon SAM 对无服务器应用程序启用代码签名。对代码进行签名有助于确保代码在签名后未被更改，并且只有来自可信发布者的签名代码包才能在您的 Lambda 函数中运行。这有助于组织摆脱在部署管线中构建网关守卫组件的负担。

有关代码签名的更多信息，请参阅《Amazon Lambda 开发人员指南》中的配置 Lambda 函数的代码签名。

在为无服务器应用程序配置代码签名之前，必须使用 S Amazon igner 创建签名配置文件。您可以将此签名配置文件用于以下任务：

1. 创建代码签名配置 – 声明 `AWS::Lambda::CodeSigningConfig` 资源以指定可信发布者的签名配置文件并设置验证检查的策略操作。您可以在与无服务器函数相同的 Amazon SAM 模板中声明此对象，也可以在不同的模板中声明此对象，也可以在 Amazon SAM 模板中声明此对象 Amazon CloudFormation 。然后，您可以使用 `AWS::Lambda::CodeSigningConfig` 资源的 Amazon 资源名称（ARN）指定该函数的 `CodeSigningConfigArn` 属性，从而为无服务器函数启用代码签名。
2. 签署代码 – 使用带 `--signing-profiles` 选项的 `sam package` 或 `sam deploy` 命令。

> **Note**
>
> 要使用 `sam package` 或 `sam deploy` 命令成功签署您的代码，必须为这些命令中使用的 Amazon S3 存储桶启用版本控制。如果您使用的是为您 Amazon SAM 创建的 Amazon S3 存储桶，则会自动启用版本控制。有关 Amazon S3 存储桶版本控制的更多信息以及在您提供的 Amazon S3 存储桶上启用版本控制的说明，请参阅《Amazon Simple Storage Service 用户指南》中的在 Amazon S3 存储桶中使用版本控制。

部署无服务器应用程序时，Lambda 会对您启用代码签名的所有函数执行验证检查。Lambda 还会对这些函数所依赖的任何层执行验证检查。有关 Lambda 验证检查的更多信息，请参阅《Amazon Lambda 开发人员指南》中的签名验证。

示例

创建签名配置文件

要创建签名配置文件，请运行以下命令：

```
aws signer put-signing-profile --platform-id "AWSLambda-SHA384-ECDSA" --profile-
name MySigningProfile
```

如果上一个命令成功，您将看到签名配置文件的 ARN 已返回。例如：

```
{
    "arn": "arn:aws:signer:us-east-1:111122223333:/signing-profiles/MySigningProfile",
    "profileVersion": "SAMPLEverx",
    "profileVersionArn": "arn:aws:signer:us-east-1:111122223333:/signing-
profiles/MySigningProfile/SAMPLEverx"
}
```

`profileVersionArn` 字段包含创建代码签名配置时要使用的 ARN。

创建代码签名配置并为函数启用代码签名

以下示例 Amazon SAM 模板声明`AWS::Lambda::CodeSigningConfig`资源并启用 Lambda 函数的代码签名。在此示例中，有一个可信配置文件，如果签名检查失败，则部署将被拒绝。

```
Resources:
  HelloWorld:
    Type: AWS::Serverless::Function
```

```
    Properties:
      CodeUri: hello_world/
      Handler: app.lambda_handler
      Runtime: python3.7
      CodeSigningConfigArn: !Ref MySignedFunctionCodeSigningConfig

  MySignedFunctionCodeSigningConfig:
    Type: AWS::Lambda::CodeSigningConfig
    Properties:
      Description: "Code Signing for MySignedLambdaFunction"
      AllowedPublishers:
        SigningProfileVersionArns:
          - MySigningProfile-profileVersionArn
      CodeSigningPolicies:
        UntrustedArtifactOnDeployment: "Enforce"
```

签署代码

您可以在打包或部署应用程序时签署代码。使用 `sam package` 或 `sam deploy` 命令指定 `--signing-profiles` 选项，如以下示例命令所示。

在打包应用程序时对函数代码进行签名：

```
sam package --signing-profiles HelloWorld=MySigningProfile --s3-bucket amzn-s3-demo-bucket --output-template-file packaged.yaml
```

在打包应用程序时，对函数代码和函数所依赖的层进行签名：

```
sam package --signing-profiles HelloWorld=MySigningProfile MyLayer=MySigningProfile --s3-bucket amzn-s3-demo-bucket --output-template-file packaged.yaml
```

对函数代码和层进行签名，然后执行部署：

```
sam deploy --signing-profiles HelloWorld=MySigningProfile MyLayer=MySigningProfile --s3-bucket amzn-s3-demo-bucket --template-file packaged.yaml --stack-name --region us-east-1 --capabilities CAPABILITY_IAM
```

> **Note**
>
> 要使用 `sam package` 或 `sam deploy` 命令成功签署您的代码，必须为这些命令中使用的 Amazon S3 存储桶启用版本控制。如果您使用的是为您 Amazon SAM 创建的 Amazon S3 存

储桶，则会自动启用版本控制。有关 Amazon S3 存储桶版本控制的更多信息以及在您提供的 Amazon S3 存储桶上启用版本控制的说明，请参阅《Amazon Simple Storage Service 用户指南》中的在 Amazon S3 存储桶中使用版本控制。

通过 `sam deploy --guided` 提供签名配置文件

当您使用配置了代码签名的无服务器应用程序运行该`sam deploy --guided`命令时， Amazon SAM 会提示您提供用于代码签名的签名配置文件。有关`sam deploy --guided`提示的更多信息，请参阅sam deploy中的 Amazon SAM CLI 命令参考。

验证 Amazon SAM 模板文件

使用 `sam validate` 验证模板。目前，此命令可验证所提供的模板是否为有效的 JSON /YAML。和大多数人一样 Amazon SAM CLI 命令，默认情况下，它会在当前工作目录中查找`template.[yaml|yml]`文件。您可以使用 `-t` 或 `--template` 选项指定其他模板文件/位置。

示例：

```
$ sam validate
<path-to-template>/template.yaml is a valid SAM Template
```

> **Note**
>
> 该`sam validate`命令需要配置 Amazon 凭据。有关更多信息，请参阅 正在配置 Amazon SAM CLI。

使用构建您的应用程序 Amazon SAM

将基础设施即代码 (IaC) 添加到 Amazon SAM 模板后，就可以开始使用sam build命令构建应用程序了。此命令根据应用程序项目目录中的文件（即 Amazon SAM 模板文件、应用程序代码以及任何适用的语言特定文件和依赖项）创建构建构件。这些构建工件为您的无服务器应用程序做好准备，以备应用程序开发的后续步骤，例如本地测试和部署到 Amazon 云端。测试和部署都使用构建构件作为输入。

您可以使用 sam build 来构建整个无服务器应用程序。此外，您可以创建自定义构建，例如具有特定函数、层或自定义运行时的构建。要详细了解您使用 sam build 的方式和原因，请参阅本节中的主题。有关使用该 `sam build` 命令的简介，请参见 建筑简介 Amazon SAM。

主题

- 建筑简介 Amazon SAM
- 默认版本使用 Amazon SAM
- 使用自定义构建 Amazon SAM

建筑简介 Amazon SAM

使用 Amazon Serverless Application Model 命令行界面 (Amazon SAM CLI) `sam build` 命令让您的无服务器应用程序为开发工作流程中的后续步骤做好准备，例如本地测试或部署到 Amazon Web Services 云。此命令会创建 `.aws-sam` 目录，该目录将以 `sam local` 和 `sam deploy` 所要求的格式和位置来编排应用程序的结构。

- 有关简介 Amazon SAM CLI，请参阅 那是什么 Amazon SAMCLI？。
- 有关 `sam build` 命令选项的列表，请参阅 sam build。
- 有关在典型开发工作流程中使用 `sam build` 的示例，请参阅第 2 步：构建应用程序。

> **Note**
>
> 使用 `sam build` 时，您需要从开发计算机上无服务器应用程序的基本组件开始。这包括 Amazon SAM 模板、 Amazon Lambda 函数代码以及任何特定于语言的文件和依赖关系。要了解更多信息，请参阅 在中创建您的应用程序 Amazon SAM。

主题

- 使用 sam build 构建应用程序
- 本地测试和部署
- 最佳实践
- sam build 的选项
- 故障排除
- 示例
- 了解更多

使用 sam build 构建应用程序

使用 `sam build` 之前，请考虑配置以下各项：

1. Lambda 函数和层 - `sam build` 命令可以构建 Lambda 函数和层。要了解有关 Lambda 层的更多信息，请参阅 在中构建 Lambda 图层 Amazon SAM。
2. Lambda 运行时 – 运行时提供在调用时在执行环境中运行函数的语言特定环境。您可以配置本机运行时和自定义运行时。
   a. 本机运行时 - 在受支持的 Lambda 运行时系统中编写 Lambda 函数，并构建函数以在 Amazon Web Services 云中使用原生 Lambda 运行时。
   b. 自定义运行时 — 使用任何编程语言创作 Lambda 函数，并使用中定义的自定义流程构建运行时 makefile 或第三方生成器，例如 esbuild。 要了解更多信息，请参阅在中使用自定义运行时构建 Lambda 函数 Amazon SAM。
3. Lambda 包类型 - Lambda 函数可以打包成以下 Lambda 部署包类型：
   a. .zip 文件归档 – 包括您的应用程序代码及其依赖项。
   b. 容器印象 – 包括基本操作系统、运行时系统、Lambda 扩展、应用程序代码及其依赖项。

可在使用 `sam init` 初始化应用程序时配置这些应用程序设置。

- 要了解有关使用 `sam init` 的更多信息，请参阅 在中创建您的应用程序 Amazon SAM。
- 要了解有关在应用程序中配置这些设置的更多信息，请参阅 默认版本使用 Amazon SAM。

构建应用程序

1. cd 到项目的根目录。此位置与您的 Amazon SAM 模板相同。

```
$ cd sam-app
```

2. 运行以下命令：

```
sam-app $ sam build <arguments> <options>
```

> **Note**
>
> 常用选项是 `--use-container`。要了解更多信息，请参阅 在提供的容器内构建 Lambda 函数。

以下是示例 Amazon SAM CLI 输出：

```
sam-app $ sam build
Starting Build use cache
Manifest file is changed (new hash: 3298f1304...d4d421) or dependency folder (.aws-
sam/deps/4d3dfad6-a267-47a6-a6cd-e07d6fae318c) is missing for (HelloWorldFunction),
 downloading dependencies and copying/building source
Building codeuri: /Users/.../sam-app/hello_world runtime: python3.12 metadata: {}
 architecture: x86_64 functions: HelloWorldFunction
Running PythonPipBuilder:CleanUp
Running PythonPipBuilder:ResolveDependencies
Running PythonPipBuilder:CopySource
Running PythonPipBuilder:CopySource

Build Succeeded

Built Artifacts  : .aws-sam/build
Built Template   : .aws-sam/build/template.yaml

Commands you can use next
=========================
[*] Validate SAM template: sam validate
[*] Invoke Function: sam local invoke
[*] Test Function in the Cloud: sam sync --stack-name {{stack-name}} --watch
[*] Deploy: sam deploy --guided
```

3. 的 Amazon SAM CLI 创建`.aws-sam`构建目录。以下是示例：

```
.aws-sam
### build
#   ### HelloWorldFunction
#   #   ### __init__.py
#   #   ### app.py
#   #   ### requirements.txt
#   ### template.yaml
### build.toml
```

根据您的应用程序的配置方式， Amazon SAM CLI 执行以下操作：

1. 在 `.aws-sam/build` 目录中下载、安装和组织依赖项。

2. 准备 Lambda 代码。这可能包括编译代码，创建可执行二进制文件，以及构建容器映像。
3. 将构建构件复制到 `.aws-sam` 目录中。格式因应用程序包类型而异。
   a. 对于 .zip 包类型，构件尚未压缩，因此可用于本地测试。的 Amazon SAM CLI 使用`sam deploy`时会压缩您的应用程序。
   b. 对于容器映像包类型，容器映像将在本地创建，在 `.aws-sam/build.toml` 文件中被引用。
4. 将 Amazon SAM 模板复制到`.aws-sam`目录中，并在必要时使用新的文件路径对其进行修改。

以下是构成 `.aws-sam` 目录中的构建构件的主要组件：

- 构建目录 - 包含在结构上彼此独立的 Lambda 函数和层。这使得 `.aws-sam/build` 目录中的每个函数或层都有唯一的结构。
- Amazon SAM 模板-根据构建过程中的更改使用更新的值进行修改。
- build.toml 文件 — 一个配置文件，其中包含由使用的编译设置 Amazon SAM CLI.

本地测试和部署

使用本地测试`sam local`或使用进行部署时`sam deploy`， Amazon SAM CLI 执行以下操作：

1. 它首先检查`.aws-sam`目录是否存在以及 Amazon SAM 模板是否位于该目录中。如果满足这些条件， Amazon SAM CLI 将其视为应用程序的根目录。
2. 如果不满足这些条件， Amazon SAM CLI 将 Amazon SAM 模板的原始位置视为应用程序的根目录。

开发时，如果对原始应用程序文件进行了更改，请在进行本地测试之前运行 `sam build` 以更新 `.aws-sam` 目录。

最佳实践

- 不要编辑 `.aws-sam/build` 目录下的任何代码。而是应更新项目文件夹中的原始源代码，然后运行 `sam build` 以更新 `.aws-sam/build` 目录。
- 修改原始文件时，运行 `sam build` 以更新 `.aws-sam/build` 目录。
- 你可能想要 Amazon SAM CLI 引用项目的原始根目录而不是`.aws-sam`目录，例如在开发和测试时`sam local`。删除`.aws-sam`目录或`.aws-sam`目录中的 Amazon SAM 模板以获得 Amazon SAM CLI 将您的原始项目目录识别为根项目目录。准备就绪后，再次运行 `sam build` 以创建 `.aws-sam` 目录。

- 运行 `sam build` 时，`.aws-sam/build` 目录每次都会被覆盖。`.aws-sam` 目录不会被覆盖。如果您想要存储文件（例如日志），请将其存储在 `.aws-sam` 中，以防止它们被覆盖。

sam build 的选项

构建单个资源

提供资源的逻辑 ID，以仅构建该资源。以下是示例：

```
$ sam build HelloWorldFunction
```

要构建嵌套应用程序或堆栈的资源，请使用格式 *<stack-logical-id>*/*<resource-logical-id>* 提供应用程序或堆栈逻辑 ID 和资源逻辑 ID：

```
$ sam build MyNestedStack/MyFunction
```

在提供的容器内构建 Lambda 函数

通过 `--use-container` 选项可以下载容器映像，并使用下载的映像来构建 Lambda 函数。然后在 `.aws-sam/build.toml` 文件中引用本地容器。

此选项需要 Docker 待安装。有关说明，请参阅 安装 Docker。

以下是该命令的示例：

```
$ sam build --use-container
```

您可以通过 `--build-image` 选项指定要使用的容器映像。以下是示例：

```
$ sam build --use-container --build-image amazon/aws-sam-cli-build-image-nodejs20.x
```

要指定用于单个函数的容器映像，请提供函数逻辑 ID。以下是示例：

```
$ sam build --use-container --build-image Function1=amazon/aws-sam-cli-build-image-
python3.12
```

传递环境变量给构建容器

使用 `--container-env-var` 传递环境变量给构建容器。以下是示例：

```
$ sam build --use-container --container-env-var Function1.GITHUB_TOKEN=<token1> --
container-env-var GLOBAL_ENV_VAR=<global-token>
```

要传递文件中的环境变量，请使用 `--container-env-var-file` 选项。以下是示例：

```
$ sam build --use-container --container-env-var-file <env.json>
```

`env.json` 文件示例：

```
{
  "MyFunction1": {
    "GITHUB_TOKEN": "TOKEN1"
  },
  "MyFunction2": {
    "GITHUB_TOKEN": "TOKEN2"
  }
}
```

加快构建包含多个函数的应用程序

当您在具有多种功能的应用程序`sam build`上运行时， Amazon SAM CLI 逐个构建每个函数。要加快构建过程，请使用 `--parallel` 选项。这样可以同时构建所有函数和层。

以下是该命令的示例：

```
$ sam build —-parallel
```

通过在源文件夹中构建项目来加快构建时间

对于受支持的运行时和构建方法，您可以使用 `--build-in-source` 选项直接在源文件夹中生成项目。默认情况下， Amazon SAM CLI 在临时目录中构建，其中包括复制源代码和项目文件。随`--build-in-source`着， Amazon SAM CLI 直接在源文件夹中构建，无需将文件复制到临时目录，从而加快构建过程。

有关支持的运行时和构建方法的列表，请参阅 `--build-in-source`。

故障排除

要排除故障 Amazon SAM CLI，请参阅 Amazon SAM CLI 故障排除。

示例

构建使用原生运行时和 .zip 包类型的应用程序

对于本示例，请参阅 教程：使用以下命令部署 Hello World 应用程序 Amazon SAM。

构建使用原生运行时和映像包类型的应用程序

首先，运行 `sam init` 以初始化新的应用程序。在交互式流程中，选择 `Image` 包类型。以下是示例：

```
$ sam init
...
Which template source would you like to use?
        1 - AWS Quick Start Templates
        2 - Custom Template Location
Choice: 1

Choose an AWS Quick Start application template
        1 - Hello World Example
        2 - Multi-step workflow
        3 - Serverless API
        4 - Scheduled task
        5 - Standalone function
        6 - Data processing
        7 - Hello World Example With Powertools
        8 - Infrastructure event management
        9 - Serverless Connector Hello World Example
        10 - Multi-step workflow with Connectors
        11 - Lambda EFS example
        12 - DynamoDB Example
        13 - Machine Learning
Template: 1

Use the most popular runtime and package type? (Python and zip) [y/N]: ENTER

Which runtime would you like to use?
        ...
        10 - java8
        11 - nodejs20.x
        12 - nodejs18.x
        13 - nodejs16.x
        ...
Runtime: 12
```

```
What package type would you like to use?
        1 - Zip
        2 - Image
Package type: 2

Based on your selections, the only dependency manager available is npm.
We will proceed copying the template using npm.

Would you like to enable X-Ray tracing on the function(s) in your application?  [y/
N]: ENTER

Would you like to enable monitoring using CloudWatch Application Insights?
For more info, please view https://docs.aws.amazon.com/AmazonCloudWatch/latest/
monitoring/cloudwatch-application-insights.html [y/N]: ENTER

Project name [sam-app]: ENTER

Cloning from https://github.com/aws/aws-sam-cli-app-templates (process may take a
 moment)

    -----------------------
    Generating application:
    -----------------------
    Name: sam-app
    Base Image: amazon/nodejs18.x-base
    Architectures: x86_64
    Dependency Manager: npm
    Output Directory: .
    Configuration file: sam-app/samconfig.toml

    Next steps can be found in the README file at sam-app/README.md

...
```

的 Amazon SAM CLI 初始化应用程序并创建以下项目目录：

```
sam-app
### README.md
### events
#   ### event.json
### hello-world
#   ### Dockerfile
#   ### app.mjs
```

```
#   ### package.json
#   ### tests
#       ### unit
#           ### test-handler.mjs
### samconfig.toml
### template.yaml
```

然后，运行 sam build 以构建应用程序：

```
sam-app $ sam build
Building codeuri: /Users/.../build-demo/sam-app runtime: None metadata: {'DockerTag':
 'nodejs18.x-v1', 'DockerContext': '/Users/.../build-demo/sam-app/hello-world',
 'Dockerfile': 'Dockerfile'} architecture: arm64 functions: HelloWorldFunction
Building image for HelloWorldFunction function
Setting DockerBuildArgs: {} for HelloWorldFunction function
Step 1/4 : FROM public.ecr.aws/lambda/nodejs:18
 ---> f5b68038c080
Step 2/4 : COPY app.mjs package*.json ./
 ---> Using cache
 ---> 834e565aae80
Step 3/4 : RUN npm install
 ---> Using cache
 ---> 31c2209dd7b5
Step 4/4 : CMD ["app.lambdaHandler"]
 ---> Using cache
 ---> 2ce2a438e89d
Successfully built 2ce2a438e89d
Successfully tagged helloworldfunction:nodejs18.x-v1

Build Succeeded

Built Artifacts  : .aws-sam/build
Built Template   : .aws-sam/build/template.yaml

Commands you can use next
=========================
[*] Validate SAM template: sam validate
[*] Invoke Function: sam local invoke
[*] Test Function in the Cloud: sam sync --stack-name {{stack-name}} --watch
[*] Deploy: sam deploy --guided
```

构建包含已编译编程语言的应用程序

在此示例中，我们使用以下命令构建包含 Lambda 函数的应用程序 Go 运行时间。

首先，我们使用初始化一个新的应用程序`sam init`，并将我们的应用程序配置为使用 Go:

```
$ sam init

...

Which template source would you like to use?
        1 - AWS Quick Start Templates
        2 - Custom Template Location
Choice: 1

Choose an AWS Quick Start application template
        1 - Hello World Example
        2 - Multi-step workflow
        3 - Serverless API
        ...
Template: 1

Use the most popular runtime and package type? (Python and zip) [y/N]: ENTER

Which runtime would you like to use?
        ...
        4 - dotnetcore3.1
        5 - go1.x
        6 - go (provided.al2)
        ...
Runtime: 5

What package type would you like to use?
        1 - Zip
        2 - Image
Package type: 1

Based on your selections, the only dependency manager available is mod.
We will proceed copying the template using mod.

Would you like to enable X-Ray tracing on the function(s) in your application?  [y/
N]: ENTER

Would you like to enable monitoring using CloudWatch Application Insights?
```

```
For more info, please view https://docs.aws.amazon.com/AmazonCloudWatch/latest/
monitoring/cloudwatch-application-insights.html [y/N]: ENTER

Project name [sam-app]: ENTER

Cloning from https://github.com/aws/aws-sam-cli-app-templates (process may take a
 moment)

    -----------------------
    Generating application:
    -----------------------
    Name: sam-app
    Runtime: go1.x
    Architectures: x86_64
    Dependency Manager: mod
    Application Template: hello-world
    Output Directory: .
    Configuration file: sam-app/samconfig.toml

    Next steps can be found in the README file at sam-app-go/README.md

...
```

的 Amazon SAM CLI 初始化应用程序。下面是应用程序目录结构的示例：

```
sam-app
### Makefile
### README.md
### events
#   ### event.json
### hello-world
#   ### go.mod
#   ### go.sum
#   ### main.go
#   ### main_test.go
### samconfig.toml
### template.yaml
```

引用 `README.md` 文件以明确此应用程序的要求。

```
...
## Requirements
* AWS CLI already configured with Administrator permission
```

```
* [Docker installed](https://www.docker.com/community-edition)
* [Golang](https://golang.org)
* SAM CLI - [Install the SAM CLI](https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/serverless-sam-cli-install.html)
...
```

然后，运行 `sam local invoke` 以测试函数。此后此命令出错 Go 未安装在我们的本地计算机上：

```
sam-app $ sam local invoke
Invoking hello-world (go1.x)
Local image was not found.
Removing rapid images for repo public.ecr.aws/sam/emulation-go1.x
Building
 image.....................................................................................
Using local image: public.ecr.aws/lambda/go:1-rapid-x86_64.

Mounting /Users/.../Playground/build/sam-app/hello-world as /var/task:ro,delegated
 inside runtime container
START RequestId: c6c5eddf-042b-4e1e-ba66-745f7c86dd31 Version: $LATEST
fork/exec /var/task/hello-world: no such file or directory: PathError
null
END RequestId: c6c5eddf-042b-4e1e-ba66-745f7c86dd31
REPORT RequestId: c6c5eddf-042b-4e1e-ba66-745f7c86dd31  Init Duration: 0.88 ms
 Duration: 175.75 ms Billed Duration: 176 ms Memory Size: 128 MB     Max Memory Used:
 128 MB
{"errorMessage":"fork/exec /var/task/hello-world: no such file or
 directory","errorType":"PathError"}%
```

然后，运行 `sam build` 以构建应用程序。从那以后我们遇到了错误 Go 未安装在我们的本地计算机上：

```
sam-app $ sam build
Starting Build use cache
Cache is invalid, running build and copying resources for following functions
 (HelloWorldFunction)
Building codeuri: /Users/.../Playground/build/sam-app/hello-world runtime: go1.x
 metadata: {} architecture: x86_64 functions: HelloWorldFunction

Build Failed
Error: GoModulesBuilder:Resolver - Path resolution for runtime: go1.x of binary: go was
 not successful
```

虽然我们可以通过配置本地计算机来正确构建函数，但我们改为将 `--use-container` 选项与 `sam build` 结合使用。的 Amazon SAM CLI 下载容器镜像，使用原生镜像构建我们的函数 GoModulesBuilder，然后将生成的二进制文件复制到我们的`.aws-sam/build/HelloWorldFunction`目录中。

```
sam-app $ sam build --use-container
Starting Build use cache
Starting Build inside a container
Cache is invalid, running build and copying resources for following functions
 (HelloWorldFunction)
Building codeuri: /Users/.../build/sam-app/hello-world runtime: go1.x metadata: {}
 architecture: x86_64 functions: HelloWorldFunction

Fetching public.ecr.aws/sam/build-go1.x:latest-x86_64 Docker container
 image.....................................................................................
Mounting /Users/.../build/sam-app/hello-world as /tmp/samcli/source:ro,delegated inside
 runtime container
Running GoModulesBuilder:Build

Build Succeeded

Built Artifacts  : .aws-sam/build
Built Template   : .aws-sam/build/template.yaml

Commands you can use next
=========================
[*] Validate SAM template: sam validate
[*] Invoke Function: sam local invoke
[*] Test Function in the Cloud: sam sync --stack-name {{stack-name}} --watch
[*] Deploy: sam deploy --guided
```

以下是 `.aws-sam` 目录的示例：

```
.aws-sam
### build
#   ### HelloWorldFunction
#   #   ### hello-world
#   ### template.yaml
### build.toml
### cache
#   ### c860d011-4147-4010-addb-2eaa289f4d95
#       ### hello-world
```

```
### deps
```

然后，运行 `sam local invoke`。这样就成功调用了函数：

```
sam-app $ sam local invoke
Invoking hello-world (go1.x)
Local image is up-to-date
Using local image: public.ecr.aws/lambda/go:1-rapid-x86_64.

Mounting /Users/.../Playground/build/sam-app/.aws-sam/build/HelloWorldFunction as /var/
task:ro,delegated inside runtime container
START RequestId: cfc8ffa8-29f2-49d4-b461-45e8c7c80479 Version: $LATEST
END RequestId: cfc8ffa8-29f2-49d4-b461-45e8c7c80479
REPORT RequestId: cfc8ffa8-29f2-49d4-b461-45e8c7c80479  Init Duration: 1.20 ms
 Duration: 1782.46 ms        Billed Duration: 1783 ms        Memory Size: 128 MB
 Max Memory Used: 128 MB
{"statusCode":200,"headers":null,"multiValueHeaders":null,"body":"Hello,
 72.21.198.67\n"}%
```

了解更多

要了解有关使用 `sam build` 命令的更多信息，请参阅以下内容：

- 学习 Amazon SAM：sam build — Serverless Land “学习 Amazon SAM” 系列开启 YouTube.
- 学习 Amazon SAM | sam build | E3 — Serverless Land “学习 Amazon SAM” 系列开启 YouTube.
- Amazon SAM 构建：它如何为部署提供工件（带有 SAM S2E8 的会话）— 开启系列的会话 Amazon SAM YouTube.
- Amazon SAM 自定义构建：如何使用 Makefile 在 SAM 中自定义构建 (S2E9) — 开启系列的会话 Amazon SAM YouTube.

默认版本使用 Amazon SAM

要构建无服务器应用程序，请使用 `sam build` 命令。此命令还会收集应用程序依赖项的构建构件，并将其以适当的格式和位置放置以供后续步骤（例如本地测试、打包和部署）使用。

您可以在清单文件（例如 `requirements.txt` (Python) 或 `package.json` (Node.js)）中指定应用程序的依赖项，也可以使用函数资源的 `Layers` 属性来指定。`Layers` 属性包含 Lambda 函数所依赖的 Amazon Lambda 层资源列表。

应用程序构建构件的格式取决于每个函数的 `PackageType` 属性。此属性的选项有：

- **`Zip`** – .zip 文件归档，包括您的应用程序代码及其依赖项。如果您将代码打包为 .zip 文件存档，则必须为函数指定 Lambda 运行时。
- **`Image`** – 容器映像，除了应用程序代码及其依赖项之外，还包括基本操作系统、运行时和扩展。

有关 Lambda 包类型的更多信息，请参阅《Amazon Lambda 开发人员指南》中的 Lambda 部署包。

主题

- 创建 .zip 文件存档
- 构建容器映像
- 容器环境变量文件
- 通过在源文件夹中构建项目来加快构建时间
- 示例
- 在外部构建函数 Amazon SAM

创建 .zip 文件存档

要将无服务器应用程序构建为 .zip 文件存档，请为无服务器函数声明 `PackageType: Zip`。

Amazon SAM 针对您指定的架构构建应用程序。如果您未指定架构，则x86_64默认 Amazon SAM 使用。

如果 Lambda 函数依赖于具有本地编译程序的包，请使用 `--use-container` 标志。此标志在本地将您的函数编译到一个行为类似于 Lambda 环境的容器中，因此当您将它们部署到云端时，它们的格式是正确的。 Amazon

当您使用该`--use-container`选项时，默认情况下会从 Amazon ECR Public 中 Amazon SAM 提取容器映像。如果您想从其他存储库或特定版本的 Amazon SAM CLI 中提取容器映像，则可以使用`--build-image`选项并提供备用容器镜像的 URI。以下是使用来自特定版本的 Amazon SAM CLI 的容器镜像构建应用程序的两个示例命令：

```
# Build a Node.js 20 application using a container image for a specific version of
 Amazon SAM CLI (1.136.0)
sam build --use-container --build-image public.ecr.aws/sam/build-nodejs22.x:1.136.0
```

```
# Build a function resource using the Python 3.13 container image from a specific
 version of Amazon SAM CLI (1.136.0)(
sam build --use-container --build-image Function1=public.ecr.aws/sam/build-
python3.13:1.136.0
```

有关构建 .zip 文件存档应用程序的其他示例，请参阅本主题后面的“示例”部分。

构建容器映像

要将无服务器应用程序构建为容器映像，请为无服务器函数声明 `PackageType: Image`。您还必须使用以下条目声明 `Metadata` 资源属性：

`Dockerfile`

与 Lambda 函数关联的 Dockerfile 的名称。

`DockerContext`

Dockerfile 的位置。

`DockerTag`

（可选）应用于已构建映像的标签。

`DockerBuildArgs`

为构建设置参数。

> **Important**
>
> 的 Amazon SAM CLI 不会编辑或混淆您在参数中包含的任何信息。`DockerBuildArgs`我们强烈建议您不要使用此部分存储敏感信息，例如密码或密钥。

以下是 `Metadata` 资源属性部分的示例：

```
    Metadata:
      Dockerfile: Dockerfile
      DockerContext: ./hello_world
      DockerTag: v1
```

要下载配置了 `Image` 软件包类型的示例应用程序，请参阅教程：使用以下命令部署 Hello World 应用程序 Amazon SAM。当系统提示您要安装哪种软件包类型时，选择 `Image`。

> **Note**
>
> 如果您在 Dockerfile 中指定了多架构基础镜像，则需要为主机的架构 Amazon SAM 构建容器镜像。要针对不同的架构进行构建，请指定使用特定目标架构的基础映像。

容器环境变量文件

要为构建容器提供包含环境变量的 JSON 文件，请在 `sam build` 命令中使用 `--container-env-var-file` 参数。您可以提供适用于所有无服务器资源的单个环境变量，也可以为每种资源提供不同的环境变量。

格式

向构建容器传递环境变量的格式取决于您为资源提供的环境变量的数量。

要为所有资源提供单个环境变量，请指定如下所示的 `Parameters` 对象：

```
{
  "Parameters": {
    "GITHUB_TOKEN": "TOKEN_GLOBAL"
  }
}
```

要为每种资源提供不同的环境变量，请如下所示为每种资源指定对象：

```
{
  "MyFunction1": {
    "GITHUB_TOKEN": "TOKEN1"
  },
  "MyFunction2": {
    "GITHUB_TOKEN": "TOKEN2"
  }
}
```

将您的环境变量另存为文件，例如名为 `env.json`。以下命令使用此文件将环境变量传递到构建容器：

```
sam build --use-container --container-env-var-file env.json
```

优先级

- 您为特定资源提供的环境变量优先于所有资源的单个环境变量。
- 您在命令行中提供的环境变量优先于文件中的环境变量。

通过在源文件夹中构建项目来加快构建时间

对于受支持的运行时和构建方法，您可以使用 `--build-in-source` 选项直接在源文件夹中生成项目。默认情况下， Amazon SAM CLI 在临时目录中构建，其中包括复制源代码和项目文件。随`--build-in-source`着， Amazon SAM CLI 直接在源文件夹中构建，无需将文件复制到临时目录，从而加快构建过程。

有关支持的运行时和构建方法的列表，请参阅 `--build-in-source`。

示例

示例 1：.zip 文件存档

以下 `sam build` 命令可创建 .zip 文件存档：

```
# Build all functions and layers, and their dependencies
sam build

# Run the build process inside a Docker container that functions like a Lambda
 environment
sam build --use-container

# Build a Node.js 20 application using a container image for a specific version of
 Amazon SAM CLI (1.136.0)
sam build --use-container --build-image public.ecr.aws/sam/build-nodejs22.x:1.136.0

# Build a function resource using the Python 3.13 container image from a specific
 version of Amazon SAM CLI (1.136.0)(
sam build --use-container --build-image Function1=public.ecr.aws/sam/build-
python3.13:1.136.0

# Build and run your functions locally
sam build && sam local invoke

# For more options
sam build --help
```

示例 2：容器映像

以下 Amazon SAM 模板以容器镜像的形式构建：

```
Resources:
  HelloWorldFunction:
    Type: AWS::Serverless::Function
    Properties:
      PackageType: Image
      ImageConfig:
        Command: ["app.lambda_handler"]
    Metadata:
      Dockerfile: Dockerfile
      DockerContext: ./hello_world
      DockerTag: v1
```

以下是 Dockerfile 的示例：

```
FROM public.ecr.aws/lambda/python:3.12

COPY app.py requirements.txt ./

RUN python3.12 -m pip install -r requirements.txt

# Overwrite the command by providing a different command directly in the template.
CMD ["app.lambda_handler"]
```

示例 3：npm ci

对于 Node.js 应用程序，您可以使用 `npm ci` 代替 `npm install` 来安装依赖项。要使用 `npm ci`，请在 Lambda 函数的 `Metadata` 资源属性中的 `BuildProperties` 下指定 `UseNpmCi: True`。要使用 `npm ci`，您的应用程序必须在 Lambda 函数的 `CodeUri` 中存在 `package-lock.json` 或 `npm-shrinkwrap.json` 文件。

以下示例使用 `npm ci` 在运行 `sam build` 时安装依赖项：

```
Resources:
  HelloWorldFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: hello-world/
      Handler: app.handler
```

```
      Runtime: nodejs20.x
      Architectures:
        - x86_64
      Events:
        HelloWorld:
          Type: Api
          Properties:
            Path: /hello
            Method: get
    Metadata:
      BuildProperties:
        UseNpmCi: True
```

在外部构建函数 Amazon SAM

默认情况下，当您运行时sam build， Amazon SAM 会生成所有函数资源。其他选项包括：

- 在之外构建所有函数资源 Amazon SAM — 如果您手动或通过其他工具构建所有函数资源，sam build则不需要这样做。您可以跳过 sam build 并继续执行流程的下一步，例如执行本地测试或部署应用程序。
- 在之外构建一些函数资源 Amazon SAM — 如果您 Amazon SAM 想构建一些函数资源，同时在外部构建其他函数资源 Amazon SAM，则可以在 Amazon SAM 模板中指定这一点。

在之外构建一些函数资源 Amazon SAM

要在使用时 Amazon SAM 跳过某个函数sam build，请在 Amazon SAM 模板中配置以下内容：

1. 向函数添加 `SkipBuild: True` 元数据属性。
2. 指定您构建的函数资源的路径。

以下是将 `TestFunction` 配置为跳过的示例。它的构建资源位于 `built-resources/TestFunction.zip`。

```
TestFunction:
  Type: AWS::Serverless::Function
  Properties:
    CodeUri: built-resources/TestFunction.zip
    Handler: TimeHandler::handleRequest
    Runtime: java11
  Metadata:
```

```
    SkipBuild: True
```

现在，当你跑步时sam build， Amazon SAM 将执行以下操作：

1. Amazon SAM 将跳过配置为的函数`SkipBuild: True`。
2. Amazon SAM 将构建所有其他函数资源并将其缓存在 `.aws-sam` build 目录中。
3. 对于跳过的函数，它们在 `.aws-sam` 构建目录中的模板将自动更新，以引用您构建的函数资源的指定路径。

   以下是 `.aws-sam` 构建目录中 `TestFunction` 的缓存模板的示例：

   ```
   TestFunction:
     Type: AWS::Serverless::Function
     Properties:
       CodeUri: ../../built-resources/TestFunction.zip
       Handler: TimeHandler::handleRequest
       Runtime: java11
     Metadata:
       SkipBuild: True
   ```

使用自定义构建 Amazon SAM

您可以自定义构建以包含特定的 Lambda 函数或 Lambda 层。函数是一种资源，您可以对其调用以在 Lambda 中运行您的代码。Lambda 层允许您从 Lambda 函数中提取代码，然后可以跨多个 Lambda 函数中重复使用这些代码。当您想专注于开发和部署单个无服务器函数，且免于管理共享依赖项或资源的复杂性时，则可以选择使用特定的 Lambda 函数自定义构建。此外，您可以选择构建 Lambda 层来帮助您缩小部署包的大小，将核心函数逻辑与依赖项分开，并允许您在多个函数之间共享依赖项。

本节中的主题探讨了您可以用来构建 Lambda 函数的一些不同方法。 Amazon SAM其中包括使用客户运行时构建 Lambda 函数和构建 Lambda 层。自定义运行时允许您安装和使用开发人员指南中 Lambda 运行时中 Amazon Lambda 未列出的语言。这样，您可以创建用于运行无服务器函数和应用程序的专业执行环境。仅构建 Lambda 层（而不是构建整个应用程序）可以在几个方面使您受益。它可以帮助您缩小部署包的大小，将核心函数逻辑与依赖项分开，并允许您在多个函数之间共享依赖项。

有关函数的更多信息，请参阅《Amazon Lambda 开发人员指南》中的 Lambda 概念。

主题

- 在 esbuild 中构建 Node.js Lambda 函数 Amazon SAM

- 使用原生 AOT 编译功能构建.NET Lambda 函数 Amazon SAM
- 使用构建 Rust Lambda 函数 Cargo Lambda 在 Amazon SAM
- 在中使用自定义运行时构建 Lambda 函数 Amazon SAM
- 在中构建 Lambda 图层 Amazon SAM

在 esbuild 中构建 Node.js Lambda 函数 Amazon SAM

要生成和打包 Node.js Amazon Lambda 函数，可以使用 Amazon SAM CLI 使用 esbuild JavaScript 捆绑器。esbuild 捆绑器支持你写入的 Lambda 函数。 TypeScript

要使用 esbuild 构建 Node.js Lambda 函数，请向您的 `AWS:Serverless::Function` 资源添加一个 `Metadata` 对象并为 `BuildMethod` 指定 `esbuild`。当你运行sam build命令时， Amazon SAM 使用 esbuild 来捆绑你的 Lambda 函数代码。

元数据属性

`Metadata` 对象针对 esbuild 支持以下属性：

BuildMethod

为应用程序指定捆绑程序。`esbuild` 是唯一受支持的值。

BuildProperties

为 Lambda 函数代码指定构建属性。

`BuildProperties` 对象针对 esbuild 支持以下属性：所有属性均为可选属性。默认情况下， Amazon SAM 使用您的 Lambda 函数处理程序作为入口点。

EntryPoints

为应用程序指定入口点。

外部

指定要从构建中省略的程序包列表。有关更多信息，请参阅《外部》中的 esbuild 网站。

格式

指定应用程序中生成 JavaScript 文件的输出格式。有关更多信息，请参阅 esbuild 网站中的格式。

加载程序

指定用于加载给定文件类型的数据的配置列表。

MainFields

指定解析程序包时要尝试导入哪些 `package.json` 字段。默认值为 `main,module`。

缩小

指定是否缩小捆绑的输出代码。默认值为 `true`。

OutExtension

自定义 esbuild 生成的文件的扩展名。有关更多信息，请参阅 esbuild 网站上的向外扩展。

源映射

指定捆绑程序是否生成源映射文件。默认值为 `false`。

如果设置为 `true`，则 `NODE_OPTIONS: --enable-source-maps` 会附加到 Lambda 函数的环境变量中，生成源映射并将其包含在函数中。

或者，如果 `NODE_OPTIONS: --enable-source-maps` 包含在函数的环境变量中，则 `Sourcemap` 会自动设置为 `true`。

发生冲突时，`Sourcemap: false` 优先于 `NODE_OPTIONS: --enable-source-maps`。

> **Note**
>
> 默认情况下，Lambda 使用 Amazon Key Management Service (Amazon KMS) 加密所有静态环境变量。使用源映射时，要成功部署，您的函数的执行角色必须具有执行 `kms:Encrypt` 操作的权限。

SourcesContent

指定是否在源映射文件中包含源代码。当 `Sourcemap` 设置为 `'true'` 时，可配置此属性。

- 指定 `SourcesContent: 'true'` 包含所有源代码。
- 指定 `SourcesContent: 'false'` 排除所有源代码。这样可以缩小源映射文件的大小，缩短启动时间，从而在生产中非常有用。但是，调试器中将无法使用源代码。

默认值为 `SourcesContent: true`。

有关更多信息，请参阅 esbuild 网站中的源内容。

目标

指定目标 ECMAScript 版本。默认值为 `es2020`。

TypeScript Lambda 函数示例

以下示例 Amazon SAM 模板片段使用 esbuild 根据中的代码创建 Node.js Lambda 函数。 TypeScript `hello-world/app.ts`

```
Resources:
  HelloWorldFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: hello-world/
      Handler: app.handler
      Runtime: nodejs20.x
      Architectures:
        - x86_64
      Events:
        HelloWorld:
          Type: Api
          Properties:
            Path: /hello
            Method: get
      Environment:
        Variables:
          NODE_OPTIONS: --enable-source-maps
    Metadata:
      BuildMethod: esbuild
      BuildProperties:
        Format: esm
        Minify: false
        OutExtension:
          - .js=.mjs
        Target: "es2020"
        Sourcemap: true
        EntryPoints:
          - app.ts
        External:
          - "<package-to-exclude>"
```

使用原生 AOT 编译功能构建.NET Lambda 函数 Amazon SAM

使用 Amazon Serverless Application Model (Amazon SAM) 构建和打包您的.NET 8 Amazon Lambda 函数，利用原生 Ahead-of-Time (AOT) 编译来缩短 Amazon Lambda 冷启动时间。

主题

- .NET 8 本机 AOT 概览
- Amazon SAM 与您的.NET 8 Lambda 函数一起使用
- 安装必备组件
- 在 Amazon SAM 模板中定义 .NET 8 Lambda 函数
- 使用构建您的应用程序 Amazon SAM CLI
- 了解更多

.NET 8 本机 AOT 概览

过去，.NET Lambda 函数的冷启动时间会影响无服务器应用程序的用户体验、系统延迟和使用成本。凭借 .NET 本机 AOT 编译，您可以缩短 Lambda 函数的冷启动时间。要了解有关.NET 8 原生 AOT 的更多信息，请参阅在 D otnet 存储库中使用原生 A O GitHub T。

Amazon SAM 与您的.NET 8 Lambda 函数一起使用

执行以下操作，使用 Amazon Serverless Application Model （Amazon SAM）配置 .NET 8 Lambda 函数：

- 在开发计算机上安装必备组件。
- 在模板中定义.NET 8 Lambda 函数。 Amazon SAM
- 使用构建您的应用程序 Amazon SAM CLI.

安装必备组件

以下是所需的先决条件：

- 的 Amazon SAM CLI
- .NET Core CLI
- Amazon.Lambda.Tools .NET Core Global Tool
- Docker

安装 Amazon SAM CLI

1. 要检查你是否已有 Amazon SAM CLI 安装后，运行以下命令：

```
sam --version
```

2. 要安装 Amazon SAM CLI，请参阅 安装 Amazon SAM CLI。
3. 要升级已安装版本的 Amazon SAM CLI，请参阅 正在升级 Amazon SAM CLI。

安装 .NET Core CLI

1. 要下载和安装 .NET Core CLI，请参阅从 Microsoft 网站下载 .NET。
2. 有关 .NET Core CLI 的更多信息，请参阅《Amazon Lambda 开发人员指南》中的 .NET Core CLI。

安装 Amazon.Lambda.Tools .NET Core Global Tool

1. 运行以下命令：

```
dotnet tool install -g Amazon.Lambda.Tools
```

2. 如果您已安装该工具，请确保该工具是使用以下命令的最新版本：

```
dotnet tool update -g Amazon.Lambda.Tools
```

3. 有关 Amazon.Lambda.Tools .NET 核心全球工具的更多信息，请参阅上的.NE T CLI 存储Amazon 库扩展。 GitHub

安装 Docker

- 使用原生 AOT 进行构建，需要 Docker 待安装。有关安装说明，请参阅安装 Docker 以与 Amazon SAM CLI。

在 Amazon SAM 模板中定义 .NET 8 Lambda 函数

要定义. NET8 在您的 Amazon SAM 模板中使用 Lambda 函数，请执行以下操作：

1. 从您选择的起始目录运行以下命令：

```
sam init
```

2. 选择 `Amazon Quick Start Templates` 以选择起始模板。
3. 选择 `Hello World Example`模板。
4. 通过输入 n，选择不使用最流行的运行时和包类型。

5. 对于运行时，选择 dotnet8。
6. 对于包类型，选择 Zip。
7. 对于您的入门模板，选择 Hello World Example using native AOT。

安装 Docker

- 使用原生 AOT 进行构建，需要 Docker 待安装。有关安装说明，请参阅安装 Docker 以与 Amazon SAM CLI。

```
Resources:
HelloWorldFunction:
  Type: AWS::Serverless::Function
  Properties:
    CodeUri: ./src/HelloWorldAot/
    Handler: bootstrap
    Runtime: dotnet8
    Architectures:
      - x86_64
    Events:
      HelloWorldAot:
        Type: Api
        Properties:
          Path: /hello
          Method: get
```

> **Note**
>
> 如果的Event属性设置AWS::Serverless::Function为Api，但未指定该RestApiId属性，则 Amazon SAM 生成AWS::ApiGateway::RestApi Amazon CloudFormation 资源。

使用构建您的应用程序 Amazon SAM CLI

在项目根目录中，运行 sam build 以开始构建应用程序。如果该PublishAot属性已在您的.NET 8 项目文件中定义， Amazon SAM CLI 将使用原生 AOT 编译进行构建。要了解有关 PublishAot 属性的更多信息，请参阅 Microsoft .NET 文档中的本机 AOT 部署。

要构建您的函数， Amazon SAM CLI 调用使用 Amazon.Lambda.Tools .NET 核心全球工具的.NET Core CLI。

> **Note**
>
> 构建时，如果项目所在目录或父目录中存在 `.sln` 文件，则包含该 `.sln` 文件的目录将挂载到容器中。如果找不到 `.sln` 文件，则只会挂载项目文件夹。因此，如果您要构建多项目应用程序，请确保 `.sln` 文件位于属性中。

了解更多

有关构建 .NET 8 Lambda 函数的更多信息，请参阅 Introducing the .NET 8 runtime for Amazon Lambda。

有关 sam build 命令的参考，请参阅 sam build。

使用构建 Rust Lambda 函数 Cargo Lambda 在 Amazon SAM

此功能为预览版 Amazon SAM ，可能会发生变化。

使用 Amazon Serverless Application Model 命令行界面 (Amazon SAM CLI) 使用你的 Rust Amazon Lambda 函数。

主题

- 先决条件
- 配置 Amazon SAM 为与 Rust Lambda 函数一起使用
- 示例

先决条件

Rust language

要安装 Rust，请参阅 “安装” Rust 中的 Rust 语言网站。

Cargo Lambda

的 Amazon SAM CLI 需要安装 Cargo Lambda，的子命令 Cargo。 有关安装说明，请参阅中的 “安装” Cargo Lambda 文档。

Docker

建造和测试 Rust Lambda 函数需要 Docker。 有关安装说明，请参阅安装 Docker。

选择加入 Amazon SAM CLI 测试版功能

由于此功能为预览版，因此您必须使用以下方法之一选择加入：

1. 使用环境变量：`SAM_CLI_BETA_RUST_CARGO_LAMBDA=1`。
2. 在您的 `samconfig.toml` 文件中添加以下内容：

```
[default.build.parameters]
beta_features = true
[default.sync.parameters]
beta_features = true
```

3. 使用支持的`--beta-features`选项时使用该选项 Amazon SAM CLI 命令。例如：

```
$ sam build --beta-features
```

4. 选择选项`y`时 Amazon SAM CLI 提示您选择加入。以下是示例：

```
$ sam build
Starting Build use cache
Build method "rust-cargolambda" is a beta feature.
Please confirm if you would like to proceed
You can also enable this beta feature with "sam build --beta-features". [y/N]: y
```

配置 Amazon SAM 为与 Rust Lambda 函数一起使用

第 1 步：配置您的 Amazon SAM 模板

使用以下内容配置您的 Amazon SAM 模板：

- 二进制 – 可选。指定模板何时包含多个 Rust Lambda 函数。
- BuildMethod – `rust-cargolambda`.
- CodeUri— `Cargo.toml` 文件路径。
- 处理程序 – `bootstrap`.
- 运行时系统 – `provided.al2`.

要了解有关自定义运行时的更多信息，请参阅《Amazon Lambda 开发者指南Amazon Lambda 》中的自定义运行时。

以下是已配置 Amazon SAM 模板的示例：

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
...
Resources:
  MyFunction:
    Type: AWS::Serverless::Function
    Metadata:
      BuildMethod: rust-cargolambda
      BuildProperties: function_a
    Properties:
      CodeUri: ./rust_app
      Handler: bootstrap
      Runtime: provided.al2
...
```

第 2 步：使用 Amazon SAM CLI 使用你的 Rust Lambda 函数

使用任何 Amazon SAM CLI 用你的 Amazon SAM 模板做命令。有关更多信息，请参阅 的 Amazon SAM CLI。

示例

Hello World 示例

在此示例中，我们使用以下方法构建 Hello World 示例应用程序 Rust 作为我们的运行时间。

首先，我们使用 `sam init` 初始化新的无服务器应用程序。在交互式流程中，我们选择 Hello World 应用程序并选择 Rust 运行时系统。

```
$ sam init
...
Which template source would you like to use?
        1 - AWS Quick Start Templates
        2 - Custom Template Location
Choice: 1

Choose an AWS Quick Start application template
        1 - Hello World Example
        2 - Multi-step workflow
        3 - Serverless API
        ...
Template: 1
```

```
Use the most popular runtime and package type? (Python and zip) [y/N]: ENTER

Which runtime would you like to use?
        1 - dotnet8
        2 - dotnet6
        3 - go (provided.al2)
        ...
        18 - python3.11
        19 - python3.10
        20 - ruby3.3
        21 - ruby3.2
        22 - rust (provided.al2)
        23 - rust (provided.al2023)
Runtime: 22

Based on your selections, the only Package type available is Zip.
We will proceed to selecting the Package type as Zip.

Based on your selections, the only dependency manager available is cargo.
We will proceed copying the template using cargo.

Would you like to enable X-Ray tracing on the function(s) in your application?  [y/
N]: ENTER

Would you like to enable monitoring using CloudWatch Application Insights?
For more info, please view https://docs.aws.amazon.com/AmazonCloudWatch/latest/
monitoring/cloudwatch-application-insights.html [y/N]: ENTER

Project name [sam-app]: hello-rust

    -----------------------
    Generating application:
    -----------------------
    Name: hello-rust
    Runtime: rust (provided.al2)
    Architectures: x86_64
    Dependency Manager: cargo
    Application Template: hello-world
    Output Directory: .
    Configuration file: hello-rust/samconfig.toml

    Next steps can be found in the README file at hello-rust/README.md
```

```
Commands you can use next
=========================
[*] Create pipeline: cd hello-rust && sam pipeline init --bootstrap
[*] Validate SAM template: cd hello-rust && sam validate
[*] Test Function in the Cloud: cd hello-rust && sam sync --stack-name {stack-name} --
watch
```

以下是 Hello World 应用程序的结构：

```
hello-rust
### README.md
### events
#   ### event.json
### rust_app
#   ### Cargo.toml
#   ### src
#       ### main.rs
### samconfig.toml
### template.yaml
```

在我们的 Amazon SAM 模板中，我们的 Rust 函数定义如下：

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
...
Resources:
  HelloWorldFunction:
    Type: AWS::Serverless::Function
    Metadata:
      BuildMethod: rust-cargolambda
    Properties:
      CodeUri: ./rust_app
      Handler: bootstrap
      Runtime: provided.al2
      Architectures:
        - x86_64
      Events:
        HelloWorld:
          Type: Api
            Path: /hello
            Method: get
```

接下来，运行 `sam build` 以构建应用程序并准备部署。的 Amazon SAM CLI 创建一个`.aws-sam`目录并在那里整理我们的构建工件。我们的函数是使用构建的 Cargo Lambda 并作为可执行的二进制文件存储在`.aws-sam/build/HelloWorldFunction/bootstrap`。

> **Note**
>
> 如果您计划在 MacOS 中运行 sam local invoke 命令，则需要在调用之前构建不同的函数。要执行此操作，请使用以下命令：
>
> - SAM_BUILD_MODE=debug sam build
>
> 仅在完成本地测试时才需要此命令。在为部署构建时，不建议这样做。

```
hello-rust$ sam build
Starting Build use cache
Build method "rust-cargolambda" is a beta feature.
Please confirm if you would like to proceed
You can also enable this beta feature with "sam build --beta-features". [y/N]: y

Experimental features are enabled for this session.
Visit the docs page to learn more about the AWS Beta terms https://aws.amazon.com/
service-terms/.

Cache is invalid, running build and copying resources for following functions
 (HelloWorldFunction)
Building codeuri: /Users/.../hello-rust/rust_app runtime: provided.al2 metadata:
 {'BuildMethod': 'rust-cargolambda'} architecture: x86_64 functions: HelloWorldFunction
Running RustCargoLambdaBuilder:CargoLambdaBuild
Running RustCargoLambdaBuilder:RustCopyAndRename

Build Succeeded

Built Artifacts  : .aws-sam/build
Built Template   : .aws-sam/build/template.yaml

Commands you can use next
=========================
[*] Validate SAM template: sam validate
[*] Invoke Function: sam local invoke
[*] Test Function in the Cloud: sam sync --stack-name {{stack-name}} --watch
```

```
[*] Deploy: sam deploy --guided
```

接下来，使用 `sam deploy --guided` 部署应用程序。

```
hello-rust$ sam deploy --guided

Configuring SAM deploy
======================

        Looking for config file [samconfig.toml] :  Found
        Reading default arguments  :  Success

        Setting default arguments for 'sam deploy'
        =========================================
        Stack Name [hello-rust]: ENTER
        AWS Region [us-west-2]: ENTER
        #Shows you resources changes to be deployed and require a 'Y' to initiate
 deploy
        Confirm changes before deploy [Y/n]: ENTER
        #SAM needs permission to be able to create roles to connect to the resources in
 your template
        Allow SAM CLI IAM role creation [Y/n]: ENTER
        #Preserves the state of previously provisioned resources when an operation
 fails
        Disable rollback [y/N]: ENTER
        HelloWorldFunction may not have authorization defined, Is this okay? [y/N]: y
        Save arguments to configuration file [Y/n]: ENTER
        SAM configuration file [samconfig.toml]: ENTER
        SAM configuration environment [default]: ENTER

        Looking for resources needed for deployment:

        ...

        Uploading to hello-rust/56ba6585d80577dd82a7eaaee5945c0b  817973 / 817973
 (100.00%)

        Deploying with following values
        ===============================
        Stack name                   : hello-rust
        Region                       : us-west-2
        Confirm changeset            : True
        Disable rollback             : False
```

```
        Deployment s3 bucket         : aws-sam-cli-managed-default-samclisam-s3-demo-
bucket-1a4x26zbcdkqr
        Capabilities                 : ["CAPABILITY_IAM"]
        Parameter overrides          : {}
        Signing Profiles             : {}

Initiating deployment
=====================

        Uploading to hello-rust/a4fc54cb6ab75dd0129e4cdb564b5e89.template  1239 / 1239
 (100.00%)


Waiting for changeset to be created..

CloudFormation stack changeset
-----------------------------------------------------------------------------------------------
Operation                   LogicalResourceId          ResourceType
 Replacement
-----------------------------------------------------------------------------------------------
+ Add                       HelloWorldFunctionHelloW   AWS::Lambda::Permission    N/A

                            orldPermissionProd

...
-----------------------------------------------------------------------------------------------

Changeset created successfully. arn:aws:cloudformation:us-
west-2:012345678910:changeSet/samcli-deploy1681427201/
f0ef1563-5ab6-4b07-9361-864ca3de6ad6


Previewing CloudFormation changeset before deployment
=====================================================
Deploy this changeset? [y/N]: y

2023-04-13 13:07:17 - Waiting for stack create/update to complete

CloudFormation events from stack operations (refresh every 5.0 seconds)
-----------------------------------------------------------------------------------------------
ResourceStatus              ResourceType               LogicalResourceId
 ResourceStatusReason
-----------------------------------------------------------------------------------------------
```

```
CREATE_IN_PROGRESS         AWS::IAM::Role             HelloWorldFunctionRole     -

CREATE_IN_PROGRESS         AWS::IAM::Role             HelloWorldFunctionRole
 Resource creation
...
-----------------------------------------------------------------------------------------------------------

CloudFormation outputs from deployed stack
-----------------------------------------------------------------------------------------------------------
Outputs

-----------------------------------------------------------------------------------------------------------
Key                 HelloWorldFunctionIamRole

Description         Implicit IAM Role created for Hello World function

Value               arn:aws:iam::012345678910:role/hello-rust-
HelloWorldFunctionRole-10II2P13AUDUY

Key                 HelloWorldApi

Description         API Gateway endpoint URL for Prod stage for Hello World function

Value               https://ggdxec9le9.execute-api.us-west-2.amazonaws.com/Prod/hello/


Key                 HelloWorldFunction

Description         Hello World Lambda Function ARN

Value               arn:aws:lambda:us-west-2:012345678910:function:hello-rust-
HelloWorldFunction-
yk4HzGzYeZBj

-----------------------------------------------------------------------------------------------------------



Successfully created/updated stack - hello-rust in us-west-2
```

为了进行测试，我们可以使用 API 端点调用 Lambda 函数。

```
$ curl https://ggdxec9le9.execute-api.us-west-2.amazonaws.com/Prod/hello/
Hello World!%
```

要在本地测试函数，首先我们要确保函数的 `Architectures` 属性与本地计算机相匹配。

```
...
Resources:
  HelloWorldFunction:
    Type: AWS::Serverless::Function # More info about Function Resource:
 https://github.com/awslabs/serverless-application-model/blob/master/
versions/2016-10-31.md#awsserverlessfunction
    Metadata:
      BuildMethod: rust-cargolambda # More info about Cargo Lambda: https://github.com/
cargo-lambda/cargo-lambda
    Properties:
      CodeUri: ./rust_app   # Points to dir of Cargo.toml
      Handler: bootstrap    # Do not change, as this is the default executable name
 produced by Cargo Lambda
      Runtime: provided.al2
      Architectures:
        - arm64
...
```

由于我们在此示例中将基础设施从 `x86_64` 修改为 `arm64`，因此我们运行 `sam build` 以更新构建构件。然后运行 `sam local invoke` 在本地调用函数。

```
hello-rust$ sam local invoke
Invoking bootstrap (provided.al2)
Local image was not found.
Removing rapid images for repo public.ecr.aws/sam/emulation-provided.al2
Building
 image.....................................................................................
Using local image: public.ecr.aws/lambda/provided:al2-rapid-arm64.

Mounting /Users/.../hello-rust/.aws-sam/build/HelloWorldFunction as /var/
task:ro,delegated, inside runtime container
START RequestId: fbc55e6e-0068-45f9-9f01-8e2276597fc6 Version: $LATEST
{"statusCode":200,"body":"Hello World!"}END RequestId:
 fbc55e6e-0068-45f9-9f01-8e2276597fc6
REPORT RequestId: fbc55e6e-0068-45f9-9f01-8e2276597fc6  Init Duration: 0.68 ms
 Duration: 130.63 ms     Billed Duration: 131 ms     Memory Size: 128 MB     Max Memory
 Used: 128 MB
```

单个 Lambda 函数项目

以下是一个包含单个 Rust Lambda 函数的无服务器应用程序的示例。

项目目录结构：

```
.
### Cargo.lock
### Cargo.toml
### src
#   ### main.rs
### template.yaml
```

Amazon SAM 模板：

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
...
Resources:
  MyFunction:
    Type: AWS::Serverless::Function
    Metadata:
      BuildMethod: rust-cargolambda
    Properties:
      CodeUri: ./
      Handler: bootstrap
      Runtime: provided.al2
...
```

多个 Lambda 函数项目

以下是一个包含多个 Rust Lambda 函数的无服务器应用程序的示例。

项目目录结构：

```
.
### Cargo.lock
### Cargo.toml
### src
#   ### function_a.rs
#   ### function_b.rs
### template.yaml
```

Amazon SAM 模板：

```
AWSTemplateFormatVersion: '2010-09-09'
```

```
Transform: AWS::Serverless-2016-10-31
...
Resources:
  FunctionA:
    Type: AWS::Serverless::Function
    Metadata:
      BuildMethod: rust-cargolambda
      BuildProperties:
        Binary: function_a
    Properties:
      CodeUri: ./
      Handler: bootstrap
      Runtime: provided.al2
  FunctionB:
    Type: AWS::Serverless::Function
    Metadata:
      BuildMethod: rust-cargolambda
      BuildProperties:
        Binary: function_b
    Properties:
      CodeUri: ./
      Handler: bootstrap
      Runtime: provided.al2
```

`Cargo.toml` 文件：

```
[package]
name = "test-handler"
version = "0.1.0"
edition = "2021"

[dependencies]
lambda_runtime = "0.6.0"
serde = "1.0.136"
tokio = { version = "1", features = ["macros"] }
tracing = { version = "0.1", features = ["log"] }
tracing-subscriber = { version = "0.3", default-features = false, features = ["fmt"] }

[[bin]]
name = "function_a"
path = "src/function_a.rs"

[[bin]]
```

```
name = "function_b"
path = "src/function_b.rs"
```

在中使用自定义运行时构建 Lambda 函数 Amazon SAM

您可以使用 `sam build` 命令构建 Lambda 函数所需的自定义运行时系统。您可以通过为函数指定 `Runtime: provided` 来声明 Lambda 函数以使用自定义运行时系统。

要构建自定义运行时系统，请使用 `BuildMethod: makefile` 条目声明 `Metadata` 资源属性。您提供自定义 makefile，在其中声明包含运行时构建命令的表单 `build-`*`function-logical-id`* 的构建目标。如有必要，您的 makefile 负责编译自定义运行时系统，并将构建构件复制到工作流程中后续步骤所需的适当位置。Makefile 的位置由函数资源的 `CodeUri` 属性指定，并且必须命名为 `Makefile`。

示例

示例 1：用 Rust 编写的函数的自定义运行时系统

> **Note**
>
> 我们建议使用以下方法构建 Lambda 函数 Cargo Lambda。 要了解更多信息，请参阅使用构建 Rust Lambda 函数 Cargo Lambda 在 Amazon SAM。

以下 Amazon SAM 模板声明了一个函数，该函数为用 Rust 编写的 Lambda 函数使用自定义运行时，并指示`sam build`为构建目标执行命令。`build-HelloRustFunction`

```
Resources:
  HelloRustFunction:
    Type: AWS::Serverless::Function
    Properties:
      FunctionName: HelloRust
      Handler: bootstrap.is.real.handler
      Runtime: provided
      MemorySize: 512
      CodeUri: .
    Metadata:
      BuildMethod: makefile
```

以下 makefile 包含构建目标和将要执行的命令。请注意，`CodeUri` 属性设置为 `.`，因此 makefile 必须位于项目根目录（即与应用程序 Amazon SAM 模板文件相同的目录）。文件名必须是 `Makefile`。

```
build-HelloRustFunction:
  cargo build --release --target x86_64-unknown-linux-musl
  cp ./target/x86_64-unknown-linux-musl/release/bootstrap $(ARTIFACTS_DIR)
```

有关设置开发环境以执行前面 `makefile` 中的 `cargo build` 命令的更多信息，请参阅博客文章 Amazon Lambda的 Rust 运行时系统。

示例 2：适用于 Python3.12 的 makefile 生成器（替代使用捆绑生成器）

您可能需要使用未包含在捆绑生成器中的库或模块。此示例显示了带有 makefile 生成器的 Python3.12 运行时的 Amazon SAM 模板。

```
Resources:
  HelloWorldFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: hello_world/
      Handler: app.lambda_handler
      Runtime: python3.12
    Metadata:
      BuildMethod: makefile
```

以下 makefile 包含构建目标和将要执行的命令。请注意，`CodeUri` 属性设置为 `hello_world`，因此 makefile 必须位于 `hello_world` 子目录的根目录中，并且文件名必须为 `Makefile`。

```
build-HelloWorldFunction:
  cp *.py $(ARTIFACTS_DIR)
  cp requirements.txt $(ARTIFACTS_DIR)
  python -m pip install -r requirements.txt -t $(ARTIFACTS_DIR)
  rm -rf $(ARTIFACTS_DIR)/bin
```

在中构建 Lambda 图层 Amazon SAM

您可以使用 Amazon SAM 来构建自定义 Lambda 层。Lambda 层允许您从 Lambda 函数中提取代码，然后可以在多个 Lambda 函数中重复使用这些代码。仅构建 Lambda 层（而不是构建整个应用程序）可以在几个方面使您受益。它可以帮助您缩小部署包的大小，将核心函数逻辑与依赖项分开，并允许您在多个函数之间共享依赖项。有关层的信息，请参阅《Amazon Lambda 开发人员指南》中的 Amazon Lambda 层。

如何在中构建 Lambda 层 Amazon SAM

> **Note**
>
> 在构建 Lambda 层之前，必须先在模板中写一个 Lambda 层。 Amazon SAM 有关执行此操作的信息和示例，请参阅使用 Lambda 图层提高效率 Amazon SAM。

要构建自定义层，请在您的 Amazon Serverless Application Model (Amazon SAM) 模板文件中对其进行声明，并在`Metadata`资源属性部分中`BuildMethod`加入一个条目。`BuildMethod` 的有效值是 Amazon Lambda 运行时系统或 `makefile` 的标识符。纳入一个 `BuildArchitecture` 条目以指定您的层支持的指令集架构。`BuildArchitecture` 的有效值是 Lambda 指令集架构。

如果指定 `makefile`，请提供自定义 makefile，在其中声明包含层构建命令的表单 `build-`*`layer-logical-id`* 的构建目标。如有必要，您的 makefile 负责编译层，并将构建构件复制到工作流程中后续步骤所需的适当位置。Makefile 的位置由层资源的 `ContentUri` 属性指定，并且必须命名为 `Makefile`。

> **Note**
>
> 创建自定义图层时，需要 Amazon Lambda 依靠环境变量来查找图层代码。Lambda 运行时系统包含将您的层代码复制到的 `/opt` 目录中的路径。项目的构建构件文件夹结构必须与运行时系统的预期文件夹结构相匹配，这样才能找到自定义层代码。
> 例如，对于 Python，您可以将代码放在 `python/` 子目录中。对于 NodeJS，您可以将代码放在 `nodejs/node_modules/` 子目录中。
> 有关更多信息，请参阅《Amazon Lambda 开发人员指南》中的在层中包括库依赖项。

以下是 `Metadata` 资源属性部分的示例。

```
Metadata:
  BuildMethod: python3.12
  BuildArchitecture: arm64
```

> **Note**
>
> 如果您不包括Metadata资源属性部分，则 Amazon SAM 不构建图层。相反，它会从层资源 CodeUri 属性中指定的位置复制构建构件。有关更多信息，请参阅 AWS::Serverless::LayerVersion 资源类型中 ContentUri 属性。

当包含Metadata资源属性部分时，您可以使用sam build命令来构建层，既可以作为独立对象，也可以作为 Amazon Lambda 函数的依赖项。

- 作为独立对象。您可能只想构建层对象，例如，当您在本地测试层的代码更改并且不需要构建整个应用程序时。要独立构建层，请使用 sam build *layer-logical-id* 命令指定层资源。
- 作为 Lambda 函数的依赖项。当您在同一 Amazon SAM 模板文件的 Lambda 函数的 Layers 属性中包含层的逻辑 ID 时，该层就是该 Lambda 函数的依赖项。当该层还包括带有 BuildMethod 条目的 Metadata 资源属性部分时，您可以通过使用 sam build 命令构建整个应用程序或使用 sam build *function-logical-id* 命令指定函数资源来构建该层。

示例

模板示例 1：针对 Python 3.12 运行时环境构建一个层

以下示例 Amazon SAM 模板基于 Python 3.12 运行时环境构建了一个层。

```
Resources:
  MyLayer:
    Type: AWS::Serverless::LayerVersion
    Properties:
      ContentUri: my_layer
      CompatibleRuntimes:
        - python3.12
    Metadata:
      BuildMethod: python3.12   # Required to have Amazon SAM build this layer
```

模板示例 2：使用自定义 makefile 构建层

以下示例 Amazon SAM 模板使用自定义模板makefile来构建图层。

```
Resources:
  MyLayer:
```

```
    Type: AWS::Serverless::LayerVersion
    Properties:
      ContentUri: my_layer
      CompatibleRuntimes:
        - python3.12
    Metadata:
      BuildMethod: makefile
```

以下 `makefile` 包含构建目标和将要执行的命令。请注意，`ContentUri` 属性设置为 `my_layer`，因此 makefile 必须位于 `my_layer` 子目录的根目录中，并且文件名必须为 `Makefile`。另请注意，构建工件被复制到`python/`子目录中，以便 Amazon Lambda 能够找到层代码。

```
build-MyLayer:
  mkdir -p "$(ARTIFACTS_DIR)/python"
  cp *.py "$(ARTIFACTS_DIR)/python"
  python -m pip install -r requirements.txt -t "$(ARTIFACTS_DIR)/python"
```

> **Note**
>
> 调用`makefile`时，会触发相应的目标，并且应将工件复制到暴露的环境变量中`$ARTIFACTS_DIR`。有关更多信息，请参阅aws-lambda-builders 中的 GitHub。

sam 构建命令示例

以下 `sam build` 命令构建包含 `Metadata` 资源属性部分的层。

```
# Build the 'layer-logical-id' resource independently
$ sam build layer-logical-id

# Build the 'function-logical-id' resource and layers that this function depends on
$ sam build function-logical-id

# Build the entire application, including the layers that any function depends on
$ sam build
```

使用以下命令测试您的无服务器应用程序 Amazon SAM

在编写和构建应用程序后，您将准备好测试您的应用程序以验证其是否正常运行。使用 Amazon SAM 命令行界面 (CLI)，您可以先在本地测试您的无服务器应用程序，然后再将其上传到 Amazon 云端。测试应用程序可帮助您确认应用程序的功能、可靠性和性能，同时识别需要解决的问题（错误）。

本节提供有关测试应用程序时可以遵循的常见做法的指导。本节中的主题主要侧重于在 Amazon 云端部署之前可以进行的本地测试。部署前测试可帮助您主动发现问题，减少与部署问题相关的不必要成本。本节中的每个主题都描述了您可以执行的测试，告诉您使用该测试的好处，并包含了展示如何执行测试的示例。测试应用程序后，您就可以调试发现的任何问题了。

主题

- 使用测试简介 sam local 命令
- 使用本地调用 Lambda 函数 Amazon SAM
- 在本地运行 API Gateway Amazon SAM
- 云测试简介 sam remote test-event
- 云端测试简介 sam remote invoke
- 使用自动执行本地集成测试 Amazon SAM
- 使用生成示例事件有效负载 Amazon SAM

使用测试简介 sam local 命令

使用 Amazon Serverless Application Model 命令行界面 (Amazon SAM CLI) `sam local` 命令在本地测试您的无服务器应用程序。

有关简介 Amazon SAM CLI，请参阅 那是什么 Amazon SAMCLI？。

先决条件

要使用`sam local`，请安装 Amazon SAM CLI 通过完成以下操作：

- Amazon SAM 先决条件.
- 安装 Amazon SAM CLI.

我们建议您在使用 `sam local` 之前初步了解以下主题：

- 正在配置 Amazon SAM CLI.
- 在中创建您的应用程序 Amazon SAM.
- 建筑简介 Amazon SAM.
- 使用部署简介 Amazon SAM.

使用 sam local 命令

使用 `sam local` 命令及其任何子命令，为应用程序执行不同类型的本地测试。

```
$ sam local <subcommand>
```

要了解有关每个子命令的更多信息，请参阅以下内容：

- 简介 sam local generate-event— 生成用于本地测试 Amazon Web Services 服务 的事件。
- 简介 sam local invoke – 在本地启动 Amazon Lambda 函数的一次性调用。
- 简介 sam local start-api – 使用本地 HTTP 服务器运行 Lambda 函数。
- 简介 sam local start-lambda— 使用本地 HTTP 服务器运行您的 Lambda 函数，以便与或一起使用。Amazon CLI SDKs

使用测试简介 sam local generate-event

使用 Amazon Serverless Application Model 命令行界面 (Amazon SAM CLI) `sam local generate-event` 子命令用于生成支持 Amazon Web Services 服务的事件有效载荷样本。然后，您可以修改这些事件并将其传递给本地资源进行测试。

- 有关简介 Amazon SAM CLI，请参阅 那是什么 Amazon SAMCLI？
- 有关 `sam local generate-event` 命令选项的列表，请参阅 sam local generate-event。

事件是一个 JSON 对象，在 Amazon Web Services 服务 执行操作或任务时生成。这些事件包含特定信息，例如，已处理的数据或事件的时间戳。大多数 Amazon Web Services 服务 都会生成事件，并且每项服务的事件都采用该服务独有的格式。

由某项服务生成的事件会作为事件源传递给其他服务。例如，置于 Amazon Simple Storage Service (Amazon S3) 存储桶中的项目可以生成事件。然后，可以将此事件用作 Amazon Lambda 函数的事件源，以进一步处理数据。

使用生成的事件的格式与`sam local generate-event` Amazon 服务创建的实际事件的结构相同。您可以修改这些事件的内容，并使用它们来测试应用程序中的资源。

先决条件

要使用`sam local generate-event`，请安装 Amazon SAM CLI 通过完成以下操作：

- Amazon SAM 先决条件.
- 安装 Amazon SAM CLI.

我们建议您在使用 `sam local generate-event` 之前初步了解以下主题：

- 正在配置 Amazon SAM CLI.
- 在中创建您的应用程序 Amazon SAM.
- 建筑简介 Amazon SAM.
- 使用部署简介 Amazon SAM.

生成示例事件

使用 Amazon SAM CLI `sam local generate-event`用于生成受支持 Amazon Web Services 服务的事件的子命令。

查看支持的列表 Amazon Web Services 服务

1. 运行以下命令：

```
$ sam local generate-event
```

2. Amazon Web Services 服务 将显示支持的列表。以下是示例：

```
$ sam local generate-event
...
Commands:
  alb
  alexa-skills-kit
  alexa-smart-home
  apigateway
  appsync
  batch
  cloudformation
```

```
  ...
```

生成本地事件

1. 运行 `sam local generate-event` 并提供受支持服务的名称。这将显示可生成的事件类型的列表。以下是示例：

```
$ sam local generate-event s3

Usage: sam local generate-event s3 [OPTIONS] COMMAND [ARGS]...

Options:
  -h, --help  Show this message and exit.

Commands:
  batch-invocation  Generates an Amazon S3 Batch Operations Invocation Event
  delete            Generates an Amazon S3 Delete Event
  put               Generates an Amazon S3 Put Event
```

2. 要生成示例事件，请运行 `sam local generate-event`，并提供服务和事件类型。

```
$ sam local generate-event <service> <event>
```

以下是示例：

```
$ sam local generate-event s3 put
{
  "Records": [
    {
      "eventVersion": "2.0",
      "eventSource": "aws:s3",
      "awsRegion": "us-east-1",
      "eventTime": "1970-01-01T00:00:00.000Z",
      "eventName": "ObjectCreated:Put",
      "userIdentity": {
        "principalId": "EXAMPLE"
      },
      "requestParameters": {
        "sourceIPAddress": "127.0.0.1"
      },
      "responseElements": {
```

```
        "x-amz-request-id": "EXAMPLE123456789",
        "x-amz-id-2": "EXAMPLE123/5678abcdefghijklambdaisawesome/
mnopqrstuvwxyzABCDEFGH"
      },
      "s3": {
        "s3SchemaVersion": "1.0",
        "configurationId": "testConfigRule",
        "bucket": {
          "name": "sam-s3-demo-bucket",
          "ownerIdentity": {
            "principalId": "EXAMPLE"
          },
          "arn": "arn:aws:s3:::sam-s3-demo-bucket"
        },
        "object": {
          "key": "test/key",
          "size": 1024,
          "eTag": "0123456789abcdef0123456789abcdef",
          "sequencer": "0A1B2C3D4E5F678901"
        }
      }
    }
  ]
}
```

这些示例事件包含占位符值。您可以修改这些值以引用应用程序中的实际资源，也可以修改这些值以帮助进行本地测试。

修改示例事件

1. 您可以在命令提示符处修改示例事件。要查看选项，请运行以下命令：

```
$ sam local generate-event <service> <event> --help
```

以下是示例：

```
$ sam local generate-event s3 put --help

Usage: sam local generate-event s3 put [OPTIONS]

Options:
```

```
  --region TEXT       Specify the region name you'd like, otherwise the
                      default = us-east-1
  --partition TEXT    Specify the partition name you'd like, otherwise the
                      default = aws
  --bucket TEXT       Specify the bucket name you'd like, otherwise the
                      default = example-bucket
  --key TEXT          Specify the key name you'd like, otherwise the default =
                      test/key
  --debug             Turn on debug logging to print debug message generated
                      by AWS SAM CLI and display timestamps.
  --config-file TEXT  Configuration file containing default parameter values.
                      [default: samconfig.toml]
  --config-env TEXT   Environment name specifying default parameter values in
                      the configuration file.  [default: default]
  -h, --help          Show this message and exit.
```

2. 在命令提示符处使用这些选项中的任何一个来修改示例事件有效负载。以下是示例：

```
$ sam local generate-event s3 put--bucket sam-s3-demo-bucket

{
  "Records": [
    {
      "eventVersion": "2.0",
      "eventSource": "aws:s3",
      "awsRegion": "us-east-1",
      "eventTime": "1970-01-01T00:00:00.000Z",
      "eventName": "ObjectCreated:Put",
      "userIdentity": {
        "principalId": "EXAMPLE"
      },
      "requestParameters": {
        "sourceIPAddress": "127.0.0.1"
      },
      "responseElements": {
        "x-amz-request-id": "EXAMPLE123456789",
        "x-amz-id-2": "EXAMPLE123/5678abcdefghijklambdaisawesome/
mnopqrstuvwxyzABCDEFGH"
      },
      "s3": {
        "s3SchemaVersion": "1.0",
        "configurationId": "testConfigRule",
        "bucket": {
          "name": "sam-s3-demo-bucket",
```

```
          "ownerIdentity": {
            "principalId": "EXAMPLE"
          },
          "arn": "arn:aws:s3:::sam-s3-demo-bucket"
        },
        "object": {
          "key": "test/key",
          "size": 1024,
          "eTag": "0123456789abcdef0123456789abcdef",
          "sequencer": "0A1B2C3D4E5F678901"
        }
      }
    }
  ]
}
```

使用生成的事件进行本地测试

将生成的事件保存在本地，然后使用其他 `sam local` 子命令进行测试。

将生成的事件保存在本地

- 运行以下命令：

  ```
  $ sam local generate-event <service> <event> <event-option> > <filename.json>
  ```

  以下示例说明了如何将事件另存为项目的 `events` 文件夹中的 `s3.json` 文件。

  ```
  sam-app$ sam local generate-event s3 put --bucket amzn-s3-demo-bucket > events/
  s3.json
  ```

使用生成的事件进行本地测试

- 使用 `--event` 选项将事件与其他 `sam local` 子命令一起传递。

  以下示例说明了如何使用 `s3.json` 事件在本地调用 Lambda 函数：

  ```
  sam-app$ sam local invoke --event events/s3.json S3JsonLoggerFunction

  Invoking src/handlers/s3-json-logger.s3JsonLoggerHandler (nodejs18.x)
  ```

```
Local image is up-to-date
Using local image: public.ecr.aws/lambda/nodejs:18-rapid-x86_64.

Mounting /Users/.../sam-app/.aws-sam/build/S3JsonLoggerFunction as /var/
task:ro,delegated, inside runtime container
START RequestId: f4f45b6d-2ec6-4235-bc7b-495ec2ae0128 Version: $LATEST
END RequestId: f4f45b6d-2ec6-4235-bc7b-495ec2ae0128
REPORT RequestId: f4f45b6d-2ec6-4235-bc7b-495ec2ae0128  Init Duration: 1.23 ms
 Duration: 9371.93 ms      Billed Duration: 9372 ms        Memory Size: 128 MB
 Max Memory Used: 128 MB
```

了解更多

有关全部 `sam local generate-event` 选项的列表，请参阅 sam local generate-event。

有关使用 `sam local` 的演示，请参阅用于本地开发的Amazon SAM 。在 SAM 系列上的 Serverless Land Sessions 中@@ 测试来自本地开发环境的 Amazon Web Services 云 资源 YouTube.

使用测试简介 sam local invoke

使用 Amazon Serverless Application Model 命令行界面 (Amazon SAM CLI) `sam local invoke` 子命令用于在本地启动对 Amazon Lambda 函数的一次性调用。

- 有关简介 Amazon SAM CLI，请参阅 那是什么 Amazon SAMCLI？
- 有关 `sam local invoke` 命令选项的列表，请参阅 sam local invoke。
- 有关在典型开发工作流程中使用 `sam local invoke` 的示例，请参阅第 7 步：（可选）在本地测试应用程序。

先决条件

要使用`sam local invoke`，请安装 Amazon SAM CLI 通过完成以下操作：

- Amazon SAM 先决条件.
- 安装 Amazon SAM CLI.

我们建议您在使用 `sam local invoke` 之前初步了解以下主题：

- 正在配置 Amazon SAM CLI.

- 在中创建您的应用程序 Amazon SAM.
- 建筑简介 Amazon SAM.
- 使用部署简介 Amazon SAM.

本地调用 Lambda 函数

当你跑步时`sam local invoke`， Amazon SAM CLI 假设您当前的工作目录是项目的根目录。的 Amazon SAM CLI 将首先在`.aws-sam`子文件夹中查找`template.[yaml|yml]`文件。如果未找到，Amazon SAM CLI 将在当前工作目录中查找`template.[yaml|yml]`文件。

在本地调用 Lambda 函数

1. 从项目的根目录中，运行以下命令：

```
$ sam local invoke <options>
```

2. 如果应用程序包含多个函数，请提供函数的逻辑 ID。以下是示例：

```
$ sam local invoke HelloWorldFunction
```

3. 的 Amazon SAM CLI 使用在本地容器中构建您的函数 Docker。 然后，它会调用你的函数并输出你的函数的响应。

   以下是示例：

```
$ sam local invoke
Invoking app.lambda_handler (python3.9)
Local image is out of date and will be updated to the latest runtime. To skip this,
 pass in the parameter --skip-pull-image
Building
 image.....................................................................................
Using local image: public.ecr.aws/lambda/python:3.9-rapid-x86_64.

Mounting /Users/.../sam-app/.aws-sam/build/HelloWorldFunction as /var/
task:ro,delegated, inside runtime container
START RequestId: 64bf7e54-5509-4762-a97c-3d740498d3df Version: $LATEST
END RequestId: 64bf7e54-5509-4762-a97c-3d740498d3df
REPORT RequestId: 64bf7e54-5509-4762-a97c-3d740498d3df  Init Duration: 1.09 ms
  Duration: 608.42 ms       Billed Duration: 609 ms Memory Size: 128 MB     Max
 Memory Used: 128 MB
```

```
{"statusCode": 200, "body": "{\"message\": \"hello world\"}"}%
```

管理日志

使用 `sam local invoke` 时，Lambda 函数运行时输出（例如日志）会输出到 `stderr`，Lambda 函数结果会输出到 `stdout`。

以下是基础 Lambda 函数的示例：

```
def handler(event, context):
    print("some log") # this goes to stderr
    return "hello world" # this goes to stdout
```

您可以保存这些标准输出。以下是示例：

```
$ sam local invoke 1> stdout.log
...

$ cat stdout.log
"hello world"

$ sam local invoke 2> stderr.log
...

$ cat stderr.log
Invoking app.lambda_handler (python3.9)
Local image is up-to-date
Using local image: public.ecr.aws/lambda/python:3.9-rapid-x86_64.
Mounting /Users/.../sam-app/.aws-sam/build/HelloWorldFunction as /var/
task:ro,delegated, inside runtime container
START RequestId: 0b46e646-3bdf-4b58-8beb-242d00912c46 Version: $LATEST
some log
END RequestId: 0b46e646-3bdf-4b58-8beb-242d00912c46
REPORT RequestId: 0b46e646-3bdf-4b58-8beb-242d00912c46  Init Duration: 0.91 ms
 Duration: 589.19 ms Billed Duration: 590 ms Memory Size: 128 MB Max Memory Used: 128
 MB
```

您可以使用这些标准输出来进一步自动化您的本地开发流程。

选项

传递自定义事件以调用 Lambda 函数

要将事件传递给 Lambda 函数，请使用 `--event` 选项。以下是示例：

```
$ sam local invoke --event events/s3.json S3JsonLoggerFunction
```

您可以使用 `sam local generate-event` 子命令创建事件。要了解更多信息，请参阅 使用测试简介 sam local generate-event。

在调用 Lambda 函数时传递环境变量

如果 Lambda 函数使用环境变量，则您可以在本地测试期间使用 `--env-vars` 选项传递这些变量。这是使用已经部署在云中应用程序中的服务在本地测试 Lambda 函数的好方法。以下是示例：

```
$ sam local invoke --env-vars locals.json
```

指定模板或函数

要为指定模板 Amazon SAM CLI 要参考，请使用`--template`选项。的 Amazon SAM CLI 将只加载该 Amazon SAM 模板及其指向的资源。

要调用嵌套应用程序或堆栈的函数，请提供应用程序或堆栈的逻辑 ID 以及函数逻辑 ID。以下是示例：

```
$ sam local invoke StackLogicalId/FunctionLogicalId
```

测试你的 Lambda 函数 Terraform project

使用该`--hook-name`选项在本地测试您的 Lambda 函数 Terraform 项目。要了解更多信息，请参阅 使用 Amazon SAM CLI 和 Terraform 进行本地调试和测试。

以下是示例：

```
$ sam local invoke --hook-name terraform --beta-features
```

最佳实践

如果应用程序有无法运行 `sam build` 的 `.aws-sam` 目录，请务必在每次更新函数代码时都运行 `sam build`。然后，运行 `sam local invoke`，以在本地测试更新后的函数代码。

本地测试是部署到云中之前进行快速开发和测试的理想解决方案。但是，本地测试并不能验证所有内容，例如，不能验证云端资源之间的权限。尽可能在云端测试应用程序。我们建议使用 `sam sync` 来加快云测试工作流程。

示例

生成 Amazon API Gateway 示例事件并使用该事件在本地调用 Lambda 函数

首先，生成 API Gateway HTTP API 事件有效负载并将其保存到 `events` 文件夹中。

```
$ sam local generate-event apigateway http-api-proxy > events/apigateway_event.json
```

然后，修改 Lambda 函数以从事件中返回参数值。

```
def lambda_handler(event, context):
    print("HelloWorldFunction invoked")
    return {
        "statusCode": 200,
        "body": json.dumps({
            "message": event['queryStringParameters']['parameter2'],
        }),
    }
```

然后，在本地调用 Lambda 函数并提供自定义事件。

```
$ sam local invoke --event events/apigateway_event.json

Invoking app.lambda_handler (python3.9)
Local image is up-to-date
Using local image: public.ecr.aws/lambda/python:3.9-rapid-x86_64.

Mounting /Users/...sam-app/.aws-sam/build/HelloWorldFunction as /var/task:ro,delegated,
 inside runtime container
START RequestId: 59535d0d-3d9e-493d-8c98-6264e8e961b8 Version: $LATEST
some log
END RequestId: 59535d0d-3d9e-493d-8c98-6264e8e961b8
REPORT RequestId: 59535d0d-3d9e-493d-8c98-6264e8e961b8  Init Duration: 1.63 ms
 Duration: 564.07 ms       Billed Duration: 565 ms Memory Size: 128 MB     Max Memory
 Used: 128 MB
{"statusCode": 200, "body": "{\"message\": \"value\"}"}%
```

在本地调用 Lambda 函数时传递环境变量

此应用程序有一个 Lambda 函数，该函数使用环境变量作为 Amazon DynamoDB 表的名称。以下是 Amazon SAM 模板中定义的函数的示例：

```
AWSTemplateFormatVersion: 2010-09-09
Transform: AWS::Serverless-2016-10-31
...
Resources:
  getAllItemsFunction:
    Type: AWS::Serverless::Function
    Properties:
      Handler: src/get-all-items.getAllItemsHandler
      Description: get all items
      Policies:
        - DynamoDBReadPolicy:
            TableName: !Ref SampleTable
      Environment:
        Variables:
          SAMPLE_TABLE: !Ref SampleTable
...
```

我们要在本地测试 Lambda 函数，同时让它与云中的 DynamoDB 表进行交互。为此，我们创建环境变量文件，并将其保存在项目的根目录中，文件名为 `locals.json`。此处为`SAMPLE_TABLE`提供的值引用了云中的 DynamoDB 表。

```
{
    "getAllItemsFunction": {
        "SAMPLE_TABLE": "dev-demo-SampleTable-1U991234LD5UM98"
    }
}
```

然后，运行 `sam local invoke`，并使用 `--env-vars` 选项传入环境变量。

```
$ sam local invoke getAllItemsFunction --env-vars locals.json

Mounting /Users/...sam-app/.aws-sam/build/HelloWorldFunction as /var/task:ro,delegated,
 inside runtime container
START RequestId: 59535d0d-3d9e-493d-8c98-6264e8e961b8 Version: $LATEST
some log
END RequestId: 59535d0d-3d9e-493d-8c98-6264e8e961b8
```

```
REPORT RequestId: 59535d0d-3d9e-493d-8c98-6264e8e961b8  Init Duration: 1.63 ms
 Duration: 564.07 ms       Billed Duration: 565 ms Memory Size: 128 MB     Max Memory
 Used: 128 MB
{"statusCode":200,"body":"{}"}
```

了解更多

有关全部 `sam local invoke` 选项的列表，请参阅 sam local invoke。

有关使用 `sam local` 的演示，请参阅用于本地开发的Amazon SAM 。在 SAM 系列上的 Serverless Land Sessions 中@@ 测试来自本地开发环境的 Amazon Web Services 云 资源 YouTube.

使用测试简介 sam local start-api

使用 Amazon Serverless Application Model 命令行界面 (Amazon SAM CLI) `sam local start-api` 子命令用于在本地运行 Amazon Lambda 函数并通过本地 HTTP 服务器主机进行测试。此类测试对由 Amazon API Gateway 端点调用的 Lambda 函数很有用。

- 有关简介 Amazon SAM CLI，请参阅 那是什么 Amazon SAMCLI？
- 有关 `sam local start-api` 命令选项的列表，请参阅 sam local start-api。
- 有关在典型开发工作流程中使用 `sam local start-api` 的示例，请参阅第 7 步：（可选）在本地测试应用程序。

先决条件

要使用`sam local start-api`，请安装 Amazon SAM CLI 通过完成以下操作：

- Amazon SAM 先决条件.
- 安装 Amazon SAM CLI.

我们建议您在使用 `sam local start-api` 之前初步了解以下主题：

- 正在配置 Amazon SAM CLI.
- 在中创建您的应用程序 Amazon SAM.
- 建筑简介 Amazon SAM.
- 使用部署简介 Amazon SAM.

使用 sam local start-api

当你跑步时`sam local start-api`，Amazon SAM CLI 假设您当前的工作目录是项目的根目录。的 Amazon SAM CLI 将首先在`.aws-sam`子文件夹中查找`template.[yaml|yml]`文件。如果未找到，Amazon SAM CLI 将在当前工作目录中查找`template.[yaml|yml]`文件。

启动本地 HTTP 服务器

1. 从项目的根目录中，运行以下命令：

```
$ sam local start-api <options>
```

2. 的 Amazon SAM CLI 在本地构建您的 Lambda 函数 Docker 容器。然后，它会输出 HTTP 服务器端点的本地地址。以下是示例：

```
$ sam local start-api

Initializing the lambda functions containers.
Local image is up-to-date
Using local image: public.ecr.aws/lambda/python:3.9-rapid-x86_64.

Mounting /Users/.../sam-app/.aws-sam/build/HelloWorldFunction as /var/
task:ro,delegated, inside runtime container
Containers Initialization is done.
Mounting HelloWorldFunction at http://127.0.0.1:3000/hello [GET]
You can now browse to the above endpoints to invoke your functions. You do not
 need to restart/reload SAM CLI while working on your functions, changes will be
 reflected instantly/automatically. If you used sam build before running local
 commands, you will need to re-run sam build for the changes to be picked up. You
 only need to restart SAM CLI if you update your AWS SAM template
2023-04-12 14:41:05 WARNING: This is a development server. Do not use it in a
 production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:3000
```

3. 您可以通过浏览器或命令提示符调用 Lambda 函数。以下是示例：

```
sam-app$ curl http://127.0.0.1:3000/hello
{"message": "Hello world!"}%
```

4. 如果您更改了 Lambda 函数代码，在刷新本地 HTTP 服务器时，请考虑以下几点：

- 如果您的应用程序没有`.aws-sam`目录，并且您的函数使用解释语言，那么 Amazon SAM CLI 将通过创建新容器并托管它来自动更新您的函数。
- 如果应用程序有 `.aws-sam` 目录，则您需要运行 `sam build` 以更新函数。然后再次运行 `sam local start-api` 以托管函数。
- 如果函数使用编译语言，或者项目需要复杂的打包支持，请运行您自己的构建解决方案来更新函数。然后再次运行 `sam local start-api` 以托管函数。

使用 Lambda 授权方的 Lambda 函数

> **Note**
>
> 此功能是新增的 Amazon SAM CLI 版本 1.80.0。要升级，请参阅 正在升级 Amazon SAM CLI。

对于使用 Lambda 授权者的 Lambda 函数， Amazon SAM CLI 将在调用您的 Lambda 函数终端节点之前自动调用您的 Lambda 授权方。

以下示例说明了如何为使用 Lambda 授权方的函数启动本地 HTTP 服务器：

```
$ sam local start-api
2023-04-17 15:02:13 Attaching import module proxy for analyzing dynamic imports

AWS SAM CLI does not guarantee 100% fidelity between authorizers locally
and authorizers deployed on AWS. Any application critical behavior should
be validated thoroughly before deploying to production.

Testing application behaviour against authorizers deployed on AWS can be done using the
 sam sync command.

Mounting HelloWorldFunction at http://127.0.0.1:3000/authorized-request [GET]
You can now browse to the above endpoints to invoke your functions. You do not need
 to restart/reload SAM CLI while working on your functions, changes will be reflected
 instantly/automatically. If you used sam build before running local commands, you will
 need to re-run sam build for the changes to be picked up. You only need to restart SAM
 CLI if you update your AWS SAM template
2023-04-17 15:02:13 WARNING: This is a development server. Do not use it in a
 production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:3000
```

```
2023-04-17 15:02:13 Press CTRL+C to quit
```

当您通过本地 HTTP 服务器调用 Lambda 函数终端节点时， Amazon SAM CLI 首先调用你的 Lambda 授权器。如果授权成功， Amazon SAM CLI 将调用您的 Lambda 函数终端节点。以下是示例：

```
$ curl http://127.0.0.1:3000/authorized-request --header "header:my_token"
{"message": "from authorizer"}%

Invoking app.authorizer_handler (python3.8)
Local image is up-to-date
Using local image: public.ecr.aws/lambda/python:3.8-rapid-x86_64.

Mounting /Users/.../sam-app/... as /var/task:ro,delegated, inside runtime container
START RequestId: 38d3b472-a2c8-4ea6-9a77-9b386989bef0 Version: $LATEST
END RequestId: 38d3b472-a2c8-4ea6-9a77-9b386989bef0
REPORT RequestId: 38d3b472-a2c8-4ea6-9a77-9b386989bef0    Init Duration: 1.08 ms
 Duration: 628.26 msBilled Duration: 629 ms    Memory Size: 128 MB    Max Memory Used:
 128 MB
Invoking app.request_handler (python3.8)
Using local image: public.ecr.aws/lambda/python:3.8-rapid-x86_64.

Mounting /Users/.../sam-app/... as /var/task:ro,delegated, inside runtime container
START RequestId: fdc12255-79a3-4365-97e9-9459d06446ff Version: $LATEST
END RequestId: fdc12255-79a3-4365-97e9-9459d06446ff
REPORT RequestId: fdc12255-79a3-4365-97e9-9459d06446ff    Init Duration: 0.95 ms
 Duration: 659.13 msBilled Duration: 660 ms    Memory Size: 128 MB    Max Memory Used:
 128 MB
No Content-Type given. Defaulting to 'application/json'.
2023-04-17 15:03:03 127.0.0.1 - - [17/Apr/2023 15:03:03] "GET /authorized-request
 HTTP/1.1" 200 -
```

选项

持续重用容器以加快本地函数调用

默认情况下， Amazon SAM CLI 每次通过本地 HTTP 服务器调用函数时，都会创建一个新容器。使用 `--warm-containers` 选项可自动重用容器进行函数调用。这加快了花费的时间 Amazon SAM CLI 为您的 Lambda 函数做好本地调用的准备。您可以通过提供 `eager` 或 `lazy` 参数进一步自定义此选项。

- `eager` - 在启动时加载所有函数的容器，且并在调用之间持久存在。
- `lazy` - 仅在首次调用每个函数时加载容器。加载的容器会保持不变，以供用于其他调用。

以下是示例：

```
$ sam local start-api --warm-containers eager
```

使用 `--warm-containers` 和修改 Lambda 函数代码时：

- 如果应用程序有 `.aws-sam` 目录，请运行 `sam build` 以更新应用程序构建构件中的函数代码。
- 当检测到代码更改时， Amazon SAM CLI 自动关闭 Lambda 函数容器。
- 当你再次调用该函数时， Amazon SAM CLI 自动创建一个新容器。

指定要用于 Lambda 函数的容器映像

默认情况下， Amazon SAM CLI 使用来自亚马逊弹性容器注册表 (Amazon ECR) Registry 的 Lambda 基础镜像在本地调用您的函数。使用 `--invoke-image` 选项引用自定义容器映像。以下是示例：

```
$ sam local start-api --invoke-image public.ecr.aws/sam/emu-python3.8
```

您可以指定要与自定义容器映像一起使用的函数。以下是示例：

```
$ sam local start-api --invoke-image Function1=amazon/aws/sam-cli-emulation-image-
python3.8
```

指定模板以在本地进行测试

要为指定模板 Amazon SAM CLI 要参考，请使用`--template`选项。的 Amazon SAM CLI 将只加载该 Amazon SAM 模板及其指向的资源。以下是示例：

```
$ sam local start-api --template myTemplate.yaml
```

指定 Lambda 函数的主机开发环境

默认情况下，`sam local start-api` 子命令使用带有 IP 地址 `127.0.0.1` 的 `localhost` 来创建 HTTP 服务器。如果本地开发环境与本地计算机隔离，则可以自定义这些值。

使用 `--container-host` 选项指定主机。以下是示例：

```
$ sam local start-api --container-host host.docker.internal
```

使用 `--container-host-interface` 选项指定应与容器端口绑定的主机网络的 IP 地址。以下是示例：

```
$ sam local start-api --container-host-interface 0.0.0.0
```

最佳实践

如果应用程序有无法运行 `sam build` 的 `.aws-sam` 目录，请务必在每次更新函数代码时都运行 `sam build`。然后，运行 `sam local start-api`，以在本地测试更新后的函数代码。

本地测试是部署到云中之前进行快速开发和测试的理想解决方案。但是，本地测试并不能验证所有内容，例如，不能验证云端资源之间的权限。尽可能在云端测试应用程序。我们建议使用 `sam sync` 来加快云测试工作流程。

了解更多

有关全部 `sam local start-api` 选项的列表，请参阅 sam local start-api。

使用测试简介 sam local start-lambda

使用 Amazon SAM CLI 子命令通过`sam local start-lambda` Amazon CLI 和调用您的 Lambda 函数。 SDKs此命令启动模拟 Lambda 的本地端点。

- 有关简介 Amazon SAM CLI，请参阅 那是什么 Amazon SAMCLI？
- 有关 `sam local start-lambda` 命令选项的列表，请参阅 sam local start-lambda。

先决条件

要使用`sam local start-lambda`，请安装 Amazon SAM CLI 通过完成以下操作：

- Amazon SAM 先决条件.
- 安装 Amazon SAM CLI.

我们建议您在使用 `sam local start-lambda` 之前初步了解以下主题：

- 正在配置 Amazon SAM CLI.
- 在中创建您的应用程序 Amazon SAM.
- 建筑简介 Amazon SAM.

- 使用部署简介 Amazon SAM.

使用 sam local start-lambda

当你跑步时`sam local start-lambda`，Amazon SAM CLI 假设您当前的工作目录是项目的根目录。的 Amazon SAM CLI 将首先在`.aws-sam`子文件夹中查找`template.[yaml|yml]`文件。如果未找到，Amazon SAM CLI 将在当前工作目录中查找`template.[yaml|yml]`文件。

要使用 sam local start-lambda

1. 从项目的根目录中，运行以下命令：

```
$ sam local start-lambda <options>
```

2. 的 Amazon SAM CLI 在本地构建您的 Lambda 函数 Docker 容器。然后，它会将本地地址输出到 HTTP 服务器端点。以下是示例：

```
$ sam local start-lambda
Initializing the lambda functions containers.
Local image is up-to-date
Using local image: public.ecr.aws/lambda/python:3.9-rapid-x86_64.

Mounting /Users/.../sam-app/hello_world as /var/task:ro,delegated, inside runtime
 container
Containers Initialization is done.
Starting the Local Lambda Service. You can now invoke your Lambda Functions defined
 in your template through the endpoint.
2023-04-13 07:25:43 WARNING: This is a development server. Do not use it in a
 production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:3001
2023-04-13 07:25:43 Press CTRL+C to quit
```

3. 使用 Amazon CLI 或在本地调 SDKs 用您的 Lambda 函数。

   以下是一个使用 Amazon CLI的示例：

```
$ aws lambda invoke --function-name "HelloWorldFunction" --endpoint-
url "http://127.0.0.1:3001" --no-verify-ssl out.txt

StatusCode: 200
(END)
```

以下是使用以下内容的示例 Amazon SDK for Python:

```
import boto3
from botocore.config import Config
from botocore import UNSIGNED

lambda_client = boto3.client('lambda',
                             endpoint_url="http://127.0.0.1:3001",
                             use_ssl=False,
                             verify=False,
                             config=Config(signature_version=UNSIGNED,
                                           read_timeout=1,
                                           retries={'max_attempts': 0}
                                           )
                            )
lambda_client.invoke(FunctionName="HelloWorldFunction")
```

选项

指定模板

要为... 指定模板 Amazon SAM CLI 要参考，请使用`--template`选项。的 Amazon SAM CLI 将只加载该 Amazon SAM 模板及其指向的资源。以下是示例：

```
$ sam local start-lambda --template myTemplate.yaml
```

有关 Amazon SAM 模板的更多信息，请参阅Amazon SAM 模板解剖学。

最佳实践

如果应用程序有无法运行 `sam build` 的 `.aws-sam` 目录，请务必在每次更新函数代码时都运行 `sam build`。然后，运行 `sam local start-lambda`，以在本地测试更新后的函数代码。

本地测试是部署到云中之前进行快速开发和测试的理想解决方案。但是，本地测试并不能验证所有内容，例如，不能验证云端资源之间的权限。尽可能在云端测试应用程序。我们建议使用 `sam sync` 来加快云测试工作流程。

了解更多

有关全部 `sam local start-lambda` 选项的列表，请参阅 sam local start-lambda。

使用本地调用 Lambda 函数 Amazon SAM

在云中进行测试或部署之前本地调用 Lambda 函数可以带来多种好处。它允许您更快地测试函数的逻辑。首先本地测试可减少在云中测试或部署期间发现问题的可能性，从而帮助您避免不必要的成本。此外，本地测试使调试变得更加容易。

您可以通过使用 sam local invoke 命令并提供函数的逻辑 ID 和事件文件，在本地调用 Lambda 函数。sam local invoke 还接受 `stdin` 作为事件。有关事件的更多信息，请参阅《Amazon Lambda 开发人员指南》中的事件。有关来自不同 Amazon 服务的事件消息格式的信息，请参阅《Amazon Lambda 开发者指南》中的Amazon Lambda 与其他服务一起使用。

> **Note**
>
> 该sam local invoke命令与 Amazon Command Line Interface (Amazon CLI) 命令相对应`aws lambda invoke`。您可以使用任一命令调用 Lambda 函数。

必须在包含要调用的函数的项目目录中运行 sam local invoke 命令。

示例：

```
# Invoking function with event file
$ sam local invoke "Ratings" -e event.json

# Invoking function with event via stdin
$ echo '{"message": "Hey, are you there?" }' | sam local invoke --event - "Ratings"

# For more options
$ sam local invoke --help
```

环境变量文件

要在本地声明可覆盖模板中定义的值的环境变量，请执行以下操作：

1. 创建包含要覆盖的环境变量的 JSON 文件。
2. 使用 `--env-vars` 参数覆盖模板中定义的值。

声明环境变量

要声明可全局应用于所有资源的环境变量，请如下所示指定 `Parameters` 对象：

```
{
    "Parameters": {
        "TABLE_NAME": "localtable",
        "BUCKET_NAME": "amzn-s3-demo-bucket",
        "STAGE": "dev"
    }
}
```

要为每个资源声明不同的环境变量，请如下所示为每个资源指定对象：

```
{
    "MyFunction1": {
        "TABLE_NAME": "localtable",
        "BUCKET_NAME": "amzn-s3-demo-bucket",
    },
    "MyFunction2": {
        "TABLE_NAME": "localtable",
        "STAGE": "dev"
    }
}
```

为每个资源指定对象时，可以使用以下标识符，这些标识符按优先级从高到低的顺序列出：

1. `logical_id`
2. `function_id`
3. `function_name`
4. 完整路径标识符

您可以在单个文件中结合使用上述两种声明环境变量的方法。如果这样做，您为特定资源提供的环境变量会优先于全局环境变量。

将环境变量保存在 JSON 文件中，例如 `env.json`。

覆盖环境变量值

要用 JSON 文件中定义的变量覆盖环境变量，请将 `--env-vars` 参数与 invoke 或 start-api 命令一起使用。例如：

```
sam local invoke --env-vars env.json
```

图层

如果应用程序包含层，有关如何在本地主机上调试层的问题，请参阅使用 Lambda 图层提高效率 Amazon SAM。

了解更多

有关在本地调用函数的动手示例，请参阅完整 Amazon SAM 研讨会中的模块 2-本地运行。

在本地运行 API Gateway Amazon SAM

在本地运行 Amazon API Gateway 可以带来多种好处。例如，在本地运行 API Gateway 可以让您在部署到 Amazon 云端之前在本地测试 API 端点。如果您先在本地测试，则通常可以减少在云中的测试和开发，从而有助于降低成本。此外，在本地运行使调试变得更加容易。

要启动可用于测试 HTTP 请求/响应功能的 API Gateway 本地实例，请使用 `sam local start-api` Amazon SAM CLI 命令。此功能具有热重载功能，让您可以快速开发和迭代函数。

> **Note**
>
> 热重载是指只刷新已更改的文件，并且应用程序的状态保持不变。相比之下，实时重载是指刷新整个应用程序，并且应用程序的状态会丢失。

有关使用 `sam local start-api` 命令的说明，请参阅 使用测试简介 sam local start-api。

默认情况下， Amazon SAM 使用 Amazon Lambda 代理集成，同时支持两种`HttpApiApi`资源类型。有关`HttpApi`资源类型的代理集成的更多信息，请参阅《API Gateway 开发者指南》 APIs中的使用 HTTP 的 Amazon Lambda 代理集成。有关代理与 `Api` 资源类型集成的更多信息，请参阅《API Gateway 开发人员指南》中的理解 API Gateway Lambda 代理集成。

示例：

```
$ sam local start-api
```

Amazon SAM 自动查找 Amazon SAM 模板中已定义`HttpApi`或`Api`事件源的所有函数。然后，它将函数挂载在定义的 HTTP 路径中。

在以下 Api 示例中，对于 GET 请求，Ratings 函数将在 /ratings 中挂载 ratings.py:handler()：

```
Ratings:
  Type: AWS::Serverless::Function
  Properties:
    Handler: ratings.handler
    Runtime: python3.9
    Events:
      Api:
        Type: Api
        Properties:
          Path: /ratings
          Method: get
```

以下是 Api 响应示例：

```
// Example of a Proxy Integration response
exports.handler = (event, context, callback) => {
    callback(null, {
        statusCode: 200,
        headers: { "x-custom-header" : "my custom header value" },
        body: "hello world"
    });
}
```

如果您修改了函数的代码，请对 sam local start-api 运行 sam build 命令以检测您的更改。

环境变量文件

要在本地声明可覆盖模板中定义的值的环境变量，请执行以下操作：

1. 创建包含要覆盖的环境变量的 JSON 文件。
2. 使用 --env-vars 参数覆盖模板中定义的值。

声明环境变量

要声明可全局应用于所有资源的环境变量，请如下所示指定 Parameters 对象：

```
{
```

```
    "Parameters": {
        "TABLE_NAME": "localtable",
        "BUCKET_NAME": "amzn-s3-demo-bucket",
        "STAGE": "dev"
    }
}
```

要为每个资源声明不同的环境变量，请如下所示为每个资源指定对象：

```
{
    "MyFunction1": {
        "TABLE_NAME": "localtable",
        "BUCKET_NAME": "amzn-s3-demo-bucket",
    },
    "MyFunction2": {
        "TABLE_NAME": "localtable",
        "STAGE": "dev"
    }
}
```

为每个资源指定对象时，可以使用以下标识符，这些标识符按优先级从高到低的顺序列出：

1. `logical_id`
2. `function_id`
3. `function_name`
4. 完整路径标识符

您可以在单个文件中结合使用上述两种声明环境变量的方法。如果这样做，您为特定资源提供的环境变量会优先于全局环境变量。

将环境变量保存在 JSON 文件中，例如 `env.json`。

覆盖环境变量值

要用 JSON 文件中定义的变量覆盖环境变量，请将 `--env-vars` 参数与 invoke 或 start-api 命令一起使用。例如：

```
$ sam local start-api --env-vars env.json
```

图层

如果应用程序包含层，有关如何在本地主机上调试层的问题，请参阅使用 Lambda 图层提高效率 Amazon SAM。

云测试简介 sam remote test-event

使用 Amazon Serverless Application Model 命令行界面 (Amazon SAM CLI) `sam remote test-event` 命令访问和管理 Amazon Lambda 函数的可共享测试事件。

要了解有关可共享测试事件的更多信息，请参阅《Amazon Lambda 开发人员指南》中的可共享测试事件。

主题

- 设置 Amazon SAM CLI 要使用 sam remote test-event
- 使用 sam remote test-event 命令
- 使用可共享测试事件
- 管理可共享测试事件

先决条件

要使用`sam remote test-event`，请安装 Amazon SAM CLI 通过完成以下操作：

- Amazon SAM 先决条件.
- 安装 Amazon SAM CLI.

如果你已经有 Amazon SAM CLI 已安装，我们建议升级到最新版本的 Amazon SAM CLI 版本。要了解更多信息，请参阅 正在升级 Amazon SAM CLI。

我们建议您在使用 `sam remote test-event` 之前初步了解以下主题：

- 正在配置 Amazon SAM CLI.
- 在中创建您的应用程序 Amazon SAM.
- 建筑简介 Amazon SAM.
- 使用部署简介 Amazon SAM.

- 使用简介 sam sync 同步到 Amazon Web Services 云.

设置 Amazon SAM CLI 要使用 sam remote test-event

完成以下设置步骤即可使用 Amazon SAM CLI `sam remote test-event` 命令：

1. 配置 Amazon SAM CLI 使用你的 Amazon Web Services 账户 — Lambda 的可共享测试事件可以由用户访问和管理 Lambda 中的可共享测试事件。 Amazon Web Services 账户要配置 Amazon SAM CLI 要使用你的 Amazon Web Services 账户，请参阅正在配置 Amazon SAM CLI。
2. 为可共享的测试事件配置权限 – 要访问和管理可共享的测试事件，您必须具有适当的权限。要了解更多信息，请参阅《Amazon Lambda 开发人员指南》中的可共享测试事件。

使用 sam remote test-event 命令

的 Amazon SAM CLI `sam remote test-eventcommand` 提供了以下子命令，您可以使用这些子命令来访问和管理可共享的测试事件：

- `delete`— 从 Amazon EventBridge 架构注册表中删除可共享的测试事件。
- `get`— 从 EventBridge 架构注册表中获取可共享的测试事件。
- `list`— 列出 EventBridge 架构注册表中某个函数的现有可共享测试事件。
- `put`— 将事件从本地文件保存到 EventBridge 架构注册表。

要列出这些子命令，请使用 Amazon SAM CLI，运行以下命令：

```
$ sam remote test-event --help
```

删除可共享测试事件

您可以使用 `delete` 子命令以及以下操作来删除可共享的测试事件：

- 提供要删除的可共享测试事件的名称。
- 提供与事件关联的 Lambda 函数的可接受 ID。
- 如果您要提供 Lambda 函数逻辑 ID，则还必须提供与 Lambda 函数关联的 Amazon CloudFormation 堆栈名称。

以下是示例：

```
$ sam remote test-event delete HelloWorldFunction --stack-name sam-app --name demo-
event
```

有关与 `delete` 子命令配合使用的选项列表，请参见 sam remote test-event delete。您也可以从中运行以下命令 Amazon SAM CLI:

```
$ sam remote test-event delete --help
```

获取可共享测试事件

您可以使用`get`子命令以及以下命令从 EventBridge 架构注册表中获取可共享的测试事件：

- 提供要获取的可共享测试事件的名称。
- 提供与事件关联的 Lambda 函数的可接受 ID。
- 如果您要提供 Lambda 函数逻辑 ID，则还必须提供与 Lambda 函数关联的 Amazon CloudFormation 堆栈名称。

以下是获取名为 `demo-event` 的可共享测试事件的示例，该事件与 `sam-app` 堆栈的 `HelloWorldFunction` Lambda 函数相关联。此命令会将事件打印到您的控制台。

```
$ sam remote test-event get HelloWorldFunction --stack-name sam-app --name demo-event
```

要获取可共享的测试事件并将其保存到本地计算机，请使用 `--output-file` 选项并提供文件路径和名称。以下是将 `demo-event` 作为 `demo-event.json` 保存于当前工作目录的示例：

```
$ sam remote test-event get HelloWorldFunction --stack-name sam-app --name demo-event
 --output-file demo-event.json
```

有关与 `get` 子命令配合使用的选项列表，请参见 sam remote test-event get。您也可以从中运行以下命令 Amazon SAM CLI:

```
$ sam remote test-event get --help
```

列出可共享测试事件

您可以从架构注册表中列出特定 Lambda 函数的所有可共享测试事件。使用 `list` 子命令和以下操作：

- 提供与事件关联的 Lambda 函数的可接受 ID。
- 如果您要提供 Lambda 函数逻辑 ID，则还必须提供与 Lambda 函数关联的 Amazon CloudFormation 堆栈名称。

以下是获取与 `sam-app` 堆栈的 `HelloWorldFunction` Lambda 函数关联的所有可共享测试事件列表的示例：

```
$ sam remote test-event list HelloWorldFunction --stack-name sam-app
```

有关与 `list` 子命令配合使用的选项列表，请参见 sam remote test-event list。您也可以从中运行以下命令 Amazon SAM CLI:

```
$ sam remote test-event list --help
```

保存可共享测试事件

您可以将可共享的测试事件保存到 EventBridge 架构注册表中。使用 `put` 子命令和以下操作：

- 提供与可共享测试事件关联的 Lambda 函数的可接受的 ID。
- 提供可共享测试事件的名称。
- 为要上传的本地事件提供文件路径和名称。

以下是将本地 `demo-event.json` 事件另存为 `demo-event` 并将其与 `sam-app` 堆栈的 `HelloWorldFunction` Lambda 函数关联的示例：

```
$ sam remote test-event put HelloWorldFunction --stack-name sam-app --name demo-event
 --file demo-event.json
```

如果 EventBridge 架构注册表中存在同名的可共享测试事件，则 Amazon SAM CLI 不会覆盖它。要覆盖，请向您的命令添加 `--force` 选项。

有关与 `put` 子命令配合使用的选项列表，请参见 sam remote test-event put。您也可以从中运行以下命令 Amazon SAM CLI:

```
$ sam remote test-event put --help
```

使用可共享测试事件

使用可共享的测试事件通过命令在中 Amazon Web Services 云 测试您的 Lambda 函数。`sam remote invoke`要了解更多信息，请参阅 将可共享的测试事件传递给云端的 Lambda 函数。

管理可共享测试事件

本主题包含有关如何管理和使用可共享测试事件的示例。

获取可共享的测试事件，对其进行修改并使用

您可以从 EventBridge 架构注册表中获取可共享的测试事件，在本地对其进行修改，然后将本地测试事件与您的 Lambda 函数一起使用。 Amazon Web Services 云以下是示例：

1. 检索可共享的测试事件 – 使用 `sam remote test-event get` 子命令检索特定 Lambda 函数的可共享测试事件并将其保存在本地：

```
$ sam remote test-event get HelloWorldFunction --stack-name sam-app --name demo-event
 --output-file demo-event.json
```

2. 修改可共享的测试事件 – 使用您选择的文本编辑器修改可共享的测试事件。
3. 使用可共享的测试事件 – 使用 `sam remote invoke` 命令并通过 `--event-file` 提供事件的文件路径和名称：

```
$ sam remote invoke HelloWorldFunction --stack-name sam-app --event-file demo-
event.json
```

获取可共享的测试事件，对其进行修改，上传并使用

您可以从 EventBridge 架构注册表中获取可共享的测试事件，在本地对其进行修改，然后将其上传。然后，您可以将可共享的测试事件直接传递给 Amazon Web Services 云中的 Lambda 函数。以下是示例：

1. 检索可共享的测试事件 – 使用 `sam remote test-event get` 子命令检索特定 Lambda 函数的可共享测试事件并将其保存在本地：

```
$ sam remote test-event get HelloWorldFunction --stack-name sam-app --name demo-event
 --output-file demo-event.json
```

2. 修改可共享的测试事件 – 使用您选择的文本编辑器修改可共享的测试事件。
3. 上传可共享的测试事件-使用`sam remote test-event put`子命令将可共享的测试事件上传并保存到架构注册表。 EventBridge 在此示例中，我们使用 `--force` 选项来覆盖可共享测试的旧版本：

```
$ sam remote test-event put HelloWorldFunction --stack-name sam-app --name demo-event
 --file demo-event.json --force
```

4. 将可共享的测试事件传递给 Lambda 函数 – 使用 `sam remote invoke` 命令将可共享的测试事件直接传递给 Amazon Web Services 云中的 Lambda 函数：

```
$ sam remote invoke HelloWorldFunction --stack-name sam-app --test-event-name demo-
event
```

云端测试简介 sam remote invoke

使用 Amazon Serverless Application Model 命令行界面 (Amazon SAM CLI) `sam remote invoke` 命令与中支持的 Amazon 资源进行交互 Amazon Web Services 云。您可使用 `sam remote invoke` 调用以下资源：

- Amazon Kinesis Data Streams – 向 Kinesis Data Streams 应用程序发送数据记录。
- Amazon Lambda – 调用事件并将其传递给 Lambda 函数。
- Amazon Simple Queue Service (Amazon SQS) – 将消息发送到 Amazon SQS 队列。
- Amazon Step Functions – 调用 Step Functions 状态机以开始执行。

有关简介 Amazon SAM CLI，请参阅 那是什么 Amazon SAMCLI？

有关在典型开发工作流程中使用 `sam remote invoke` 的示例，请参阅第 5 步：在中与你的函数进行交互 Amazon Web Services 云。

主题

- 使用 sam remote invoke 命令
- 使用 sam remote invoke 命令选项
- 配置您的项目配置文件
- 示例
- 相关链接

先决条件

要使用`sam remote invoke`，请安装 Amazon SAM CLI 通过完成以下操作：

- Amazon SAM 先决条件.
- 安装 Amazon SAM CLI.

我们还建议升级到最新版本的 Amazon SAM CLI。 要了解更多信息，请参阅正在升级 Amazon SAM CLI。

我们建议您在使用 `sam remote invoke` 之前初步了解以下主题：

- 正在配置 Amazon SAM CLI.
- 在中创建您的应用程序 Amazon SAM.
- 建筑简介 Amazon SAM.
- 使用部署简介 Amazon SAM.
- 使用简介 sam sync 同步到 Amazon Web Services 云.

使用 sam remote invoke 命令

在使用此命令之前，必须将您的资源部署到 Amazon Web Services 云。

使用以下命令结构并从项目根目录运行：

```
$ sam remote invoke <arguments> <options>
```

> **Note**
>
> 此页将显示在命令提示符处提供的选项。您也可以在项目的配置文件中配置选项，而不必在命令提示符下传递这些选项。要了解更多信息，配置项目设置。

有关 `sam remote invoke` 参数和选项的说明，请参阅 sam remote invoke。

使用 Kinesis Data Streams

您可以将数据记录发送到 Kinesis Data Streams 应用程序。的 Amazon SAM CLI 将发送您的数据记录并返回分片 ID 和序列号。以下是示例：

```
$ sam remote invoke KinesisStream --stack-name kinesis-example --event hello-world

 Putting record to Kinesis data stream KinesisStream

 Auto converting value 'hello-world' into JSON '"hello-world"'. If you don't want auto-
conversion, please provide
 a JSON string as event

 {
   "ShardId": "shardId-000000000000",
   "SequenceNumber": "49646251411914806775980850790050483811301135051202232322"
 }%
```

发送数据记录

1. 提供资源 ID 值作为 Kinesis Data Streams 应用程序的参数。有关有效资源的信息 IDs，请参阅资源 ID。
2. 提供数据记录作为事件发送到 Kinesis Data Streams 应用程序。您可以使用 `--event` 选项在命令行中提供事件，也可以使用 `--event-file` 从文件中提供事件。如果您不提供活动， Amazon SAM CLI 发送一个空事件。

与 Lambda 函数一起使用

您可以在云中调用 Lambda 函数并传递空事件，或者在命令行或文件中提供事件。的 Amazon SAM CLI 将调用您的 Lambda 函数并返回其响应。以下是示例：

```
$ sam remote invoke HelloWorldFunction --stack-name sam-app

Invoking Lambda Function HelloWorldFunction
START RequestId: d5ef494b-5f45-4086-86fd-d7322fa1a1f9 Version: $LATEST
END RequestId: d5ef494b-5f45-4086-86fd-d7322fa1a1f9
REPORT RequestId: d5ef494b-5f45-4086-86fd-d7322fa1a1f9  Duration: 6.62 ms       Billed
 Duration: 7 ms     Memory Size: 128 MB     Max Memory Used: 67 MB  Init Duration:
 164.06 ms
{"statusCode":200,"body":"{\"message\":\"hello world\"}"}%
```

要调用 Lambda 函数

1. 提供资源 ID 值作为 Lambda 函数的参数。有关有效资源的信息 IDs，请参阅资源 ID。

2. 提供向 Lambda 函数发送的事件。您可以使用 `--event` 选项在命令行中提供事件，也可以使用 `--event-file` 从文件中提供事件。如果您不提供活动， Amazon SAM CLI 发送一个空事件。

配置了响应流式处理的 Lambda 函数

`sam remote invoke` 命令支持配置为流式传输响应的 Lambda 函数。您可以使用模板中的`FunctionUrlConfig`属性配置 Lambda 函数以流式传输响应。 Amazon SAM 当你使用时`sam remote invoke`， Amazon SAM CLI 将自动检测您的 Lambda 配置并通过响应流进行调用。

有关示例，请参阅调用配置为流式传输响应的 Lambda 函数。

将可共享的测试事件传递给云端的 Lambda 函数

可共享测试事件是您可与同一 Amazon Web Services 账户中的其他人共享的测试事件。要了解更多信息，请参阅《Amazon Lambda 开发人员指南》中的可共享测试事件。

访问和管理可共享测试事件

你可以使用 Amazon SAM CLI `sam remote test-event`命令访问和管理可共享的测试事件。例如，您可以使用 `sam remote test-event` 执行以下操作：

- 从 Amazon EventBridge 架构注册表中检索可共享的测试事件。
- 在本地修改可共享的测试事件并将其上传到 EventBridge 架构注册表。
- 从 EventBridge 架构注册表中删除可共享的测试事件。

要了解更多信息，请参阅云测试简介 sam remote test-event。

将可共享的测试事件传递给云中的 Lambda 函数

要将可共享的测试事件从 EventBridge 架构注册表传递到云中的 Lambda 函数，请使用`--test-event-name`选项并提供可共享测试事件的名称。以下是示例：

```
$ sam remote invoke HelloWorldFunction --stack-name sam-app --test-event-name demo-event
```

如果将可共享的测试事件保存在本地，则可以使用 `--event-file` 选项并提供本地测试事件的文件路径和名称。以下是示例：

```
$ sam remote invoke HelloWorldFunction --stack-name sam-app --event-file demo-event.json
```

将 与 Amazon SQS 结合使用

您可以向 Amazon SQS 队列发送消息。的 Amazon SAM CLI 会将返回以下内容：

- 消息 ID
- MD5 的邮件正文
- 响应元数据

以下是示例：

```
$ sam remote invoke MySqsQueue --stack-name sqs-example -event hello

Sending message to SQS queue MySqsQueue

{
  "MD5OfMessageBody": "5d41402abc4b2a76b9719d911017c592",
  "MessageId": "05c7af65-9ae8-4014-ae28-809d6d8ec652"
}%
```

发送邮件

1. 提供资源 ID 值作为 Amazon SQS 队列的参数。有关有效资源的信息 IDs，请参阅资源 ID。
2. 提供要发送给 Amazon SQS 队列的事件。您可以使用 `--event` 选项在命令行中提供事件，也可以使用 `--event-file` 从文件中提供事件。如果您不提供活动， Amazon SAM CLI 发送一个空事件。

配合使用 和 Step Functions

您可调用 Step Functions 状态机以开始执行。的 Amazon SAM CLI 将等待状态机工作流程完成并返回执行最后一步的输出。以下是示例：

```
$ sam remote invoke HelloWorldStateMachine --stack-name state-machine-example --
event '{"is_developer": true}'

Invoking Step Function HelloWorldStateMachine

"Hello Developer World"%
```

要调用状态机

1. 为 Step Functions 状态机提供一个资源 ID 值作为参数。有关有效资源的信息 IDs，请参阅资源 ID。
2. 提供要发送到状态机的事件。您可以使用 `--event` 选项在命令行中提供事件，也可以使用 `--event-file` 从文件中提供事件。如果您不提供活动， Amazon SAM CLI 发送一个空事件。

使用 sam remote invoke 命令选项

本节介绍一些可用于 `sam remote invoke` 命令的主要选项。有关选项的完整列表，请参阅 sam remote invoke。

将活动传递给您的资源

使用以下选项将事件传递给云中的资源：

- `--event` – 在命令行传递事件。
- `--event-file` – 从文件传递事件。

Lambda 示例

使用 **`--event`** 在命令行中将事件作为字符串值传递：

```
$ sam remote invoke HelloWorldFunction --stack-name sam-app --event '{"message":
 "hello!"}'

Invoking Lambda Function HelloWorldFunction


START RequestId: b992292d-1fac-4aa2-922a-c9dc5c6fceab Version: $LATEST
END RequestId: b992292d-1fac-4aa2-922a-c9dc5c6fceab
REPORT RequestId: b992292d-1fac-4aa2-922a-c9dc5c6fceab  Duration: 16.41 ms      Billed
 Duration: 17 ms  Memory Size: 128 MB     Max Memory Used: 67 MB  Init Duration: 185.96
 ms
{"statusCode":200,"body":"{\"message\":\"hello!\"}"}%
```

使用 **`--event-file`** 从文件传递事件并提供文件路径：

```
$ cat event.json
```

```
{"message": "hello from file"}%

$ sam remote invoke HelloWorldFunction --stack-name sam-app --event-file event.json


Invoking Lambda Function HelloWorldFunction


START RequestId: 3bc71f7d-153a-4b1e-8c9a-901d91b1bec9 Version: $LATEST
END RequestId: 3bc71f7d-153a-4b1e-8c9a-901d91b1bec9
REPORT RequestId: 3bc71f7d-153a-4b1e-8c9a-901d91b1bec9  Duration: 21.15 ms      Billed
 Duration: 22 ms  Memory Size: 128 MB     Max Memory Used: 67 MB
{"statusCode":200,"body":"{\"message\":\"hello from file\"}"}%
```

使用 **stdin** 传递事件：

```
$ cat event.json

{"message": "hello from file"}%

$ cat event.json | sam remote invoke HelloWorldFunction --stack-name sam-app --event-
file -

Reading event from stdin (you can also pass it from file with --event-file)

Invoking Lambda Function HelloWorldFunction

START RequestId: 85ecc902-8ad0-4a2b-a8c8-9bb4f65f5a7a Version: $LATEST
END RequestId: 85ecc902-8ad0-4a2b-a8c8-9bb4f65f5a7a
REPORT RequestId: 85ecc902-8ad0-4a2b-a8c8-9bb4f65f5a7a  Duration: 1.36 ms       Billed
 Duration: 2 ms   Memory Size: 128 MB       Max Memory Used: 67 MB
{"statusCode":200,"body":"{\"message\":\"hello from file\"}"}%
```

配置 Amazon SAM CLI 响应输出

当您使用调用支持的资源时`sam remote invoke`，Amazon SAM CLI 返回包含以下内容的响应：

- 请求元数据 – 与请求关联的元数据。这包括请求 ID 和请求开始时间。
- 资源响应 – 在云中被调用后来自资源的响应。

您可以使用该`--output`选项来配置 Amazon SAM CLI 输出响应。以下选项值可用：

- `json`— 元数据和资源响应以以下形式返回 JSON 结构。响应包含完整内容 SDK 输出。
- `text` – 元数据以文本结构返回。资源响应以资源的输出格式返回。

下面是 `json` 输出的一个示例。

```
$ sam remote invoke --stack-name sam-app --output json

Invoking Lambda Function HelloWorldFunction


{
  "ResponseMetadata": {
    "RequestId": "3bdf9a30-776d-4a90-94a6-4cccc0fc7b41",
    "HTTPStatusCode": 200,
    "HTTPHeaders": {
      "date": "Mon, 19 Jun 2023 17:15:46 GMT",
      "content-type": "application/json",
      "content-length": "57",
      "connection": "keep-alive",
      "x-amzn-requestid": "3bdf9a30-776d-4a90-94a6-4cccc0fc7b41",
      "x-amzn-remapped-content-length": "0",
      "x-amz-executed-version": "$LATEST",
      "x-amz-log-result":
 "U1RBUlQgUmVxdWVzdElkOiAzYmRmOWEzMC03NzZkLTRhOTAtOTRhNi00Y2NjYzBmYzdiNDEgVmVyc2lvbjogJExBVEVTV
      "x-amzn-trace-id":
 "root=1-64908d42-17dab270273fcc6b527dd6b8;sampled=0;lineage=2301f8dc:0"
    },
    "RetryAttempts": 0
  },
  "StatusCode": 200,
  "LogResult":
 "U1RBUlQgUmVxdWVzdElkOiAzYmRmOWEzMC03NzZkLTRhOTAtOTRhNi00Y2NjYzBmYzdiNDEgVmVyc2lvbjogJExBVEVTV
  "ExecutedVersion": "$LATEST",
  "Payload": "{\"statusCode\":200,\"body\":\"{\\\"message\\\":\\\"hello world\\\"}\"}"
}%
```

当您指定 `json` 输出时，整个响应将返回到 `stdout`。以下是示例：

```
$ sam remote invoke --stack-name sam-app --output json 1> stdout.log

Invoking Lambda Function HelloWorldFunction
```

```
$ cat stdout.log

{
  "ResponseMetadata": {
    "RequestId": "d30d280f-8188-4372-bc94-ce0f1603b6bb",
    "HTTPStatusCode": 200,
    "HTTPHeaders": {
      "date": "Mon, 19 Jun 2023 17:35:56 GMT",
      "content-type": "application/json",
      "content-length": "57",
      "connection": "keep-alive",
      "x-amzn-requestid": "d30d280f-8188-4372-bc94-ce0f1603b6bb",
      "x-amzn-remapped-content-length": "0",
      "x-amz-executed-version": "$LATEST",
      "x-amz-log-result":
 "U1RBUlQgUmVxdWVzdElkOiBkMzBkMjgwZi04MTg4LTQzNzItYmM5NC1jZTBmMTYwM2I2YmIgVmVyc2lvbjogJExBVEVTV
      "x-amzn-trace-id":
 "root=1-649091fc-771473c7778689627a6122b7;sampled=0;lineage=2301f8dc:0"
    },
    "RetryAttempts": 0
  },
  "StatusCode": 200,
  "LogResult":
 "U1RBUlQgUmVxdWVzdElkOiBkMzBkMjgwZi04MTg4LTQzNzItYmM5NC1jZTBmMTYwM2I2YmIgVmVyc2lvbjogJExBVEVTV
  "ExecutedVersion": "$LATEST",
  "Payload": "{\"statusCode\":200,\"body\":\"{\\\"message\\\":\\\"hello world\\\"}\"}"
}%
```

下面是 text 输出的一个示例。

```
$ sam remote invoke --stack-name sam-app --output text

Invoking Lambda Function HelloWorldFunction


START RequestId: 4dbacc43-1ec6-47c2-982b-9dc4620144d6 Version: $LATEST
END RequestId: 4dbacc43-1ec6-47c2-982b-9dc4620144d6
REPORT RequestId: 4dbacc43-1ec6-47c2-982b-9dc4620144d6  Duration: 9.13 ms       Billed
 Duration: 10 ms  Memory Size: 128 MB     Max Memory Used: 67 MB  Init Duration: 165.50
 ms
```

```
{"statusCode":200,"body":"{\"message\":\"hello world\"}"}%
```

当您指定 `text` 输出时，Lambda 函数运行时系统输出（例如日志）将返回到 `stderr`。Lambda 函数的有效负载将返回到 `stdout`。以下是示例：

```
$ sam remote invoke --stack-name sam-app --output text 2> stderr.log

{"statusCode":200,"body":"{\"message\":\"hello world\"}"}%



$ cat stderr.log

Invoking Lambda Function HelloWorldFunction
START RequestId: 82273c3b-aa3a-4d16-8f1c-1d2ad3ace891 Version: $LATEST
END RequestId: 82273c3b-aa3a-4d16-8f1c-1d2ad3ace891
REPORT RequestId: 82273c3b-aa3a-4d16-8f1c-1d2ad3ace891  Duration: 40.62 ms      Billed
 Duration: 41 ms  Memory Size: 128 MB     Max Memory Used: 68 MB

$ sam remote invoke --stack-name sam-app --output text 1> stdout.log

Invoking Lambda Function HelloWorldFunction



START RequestId: 74acaa9f-5b80-4a5c-b3b8-ffaccb84cbbd Version: $LATEST
END RequestId: 74acaa9f-5b80-4a5c-b3b8-ffaccb84cbbd
REPORT RequestId: 74acaa9f-5b80-4a5c-b3b8-ffaccb84cbbd  Duration: 2.31 ms       Billed
 Duration: 3 ms   Memory Size: 128 MB     Max Memory Used: 67 MB

$ cat stdout.log

{"statusCode":200,"body":"{\"message\":\"hello world\"}"}%
```

自定义 Boto3 参数

因为`sam remote invoke`， Amazon SAM CLI 利用适用于 Python 的 Amazon SDK (Boto3) 与你的云端资源进行交互。您可以使用该`--parameter`选项进行自定义 Boto3 参数。有关可自定义受支持的参数的列表，请参阅 `--parameter`。

示例

调用 Lambda 函数来验证参数值并验证权限：

```
$ sam remote invoke HelloWorldFunction --stack-name sam-app --
parameter InvocationType="DryRun"
```

在单个命令中多次使用 **--parameter** 选项以提供多个参数：

```
$ sam remote invoke HelloWorldFunction --stack-name sam-app --
parameter InvocationType="Event" --parameter LogType="None"
```

其他选项

有关 `sam remote invoke` 选项的完整列表，请参阅 sam remote invoke。

配置您的项目配置文件

要在配置文件中配置 `sam remote invoke`，请在表中使用 `remote_invoke`。以下是配置 `sam remote invoke` 命令默认值的 `samconfig.toml` 文件示例。

```
...
version =0.1

[default]
...
[default.remote_invoke.parameters]
stack_name = "cloud-app"
event = '{"message": "Hello!"}'
```

示例

有关使用的基本示例`sam remote invoke`，请参阅 C Amazon ompute 博客中的使用 Amazon SAM 远程测试 Amazon Lambda 函数。

Kinesis Data Streams 示例

基本示例

将数据记录从文件发送到 Kinesis Data Streams 应用程序。Kinesis Data Streams 应用程序是通过提供资源 ID 的 ARN 来识别的：

```
$ sam remote invoke arn:aws:kinesis:us-west-2:01234567890:stream/kinesis-example-
KinesisStream-BgnLcAey4xUQ --event-file event.json
```

将命令行中提供的事件发送到 Kinesis Data Streams 应用程序：

```
$ sam remote invoke KinesisStream --stack-name kinesis-example --event hello-world

Putting record to Kinesis data stream KinesisStream

Auto converting value 'hello-world' into JSON '"hello-world"'. If you don't want auto-
conversion, please provide
a JSON string as event

{
  "ShardId": "shardId-000000000000",
  "SequenceNumber": "49646251411914806775980903986194508740483329854174920706"
}%
```

获取 Kinesis Data Streams 应用程序的物理 ID。然后，在命令行中提供一个事件：

```
$ sam list resources --stack-name kinesis-example --output json

[
  {
    "LogicalResourceId": "KinesisStream",
    "PhysicalResourceId": "kinesis-example-KinesisStream-ZgnLcQey4xUQ"
  }
]

$ sam remote invoke kinesis-example-KinesisStream-ZgnLcQey4xUQ --event hello

Putting record to Kinesis data stream KinesisStream

Auto converting value 'hello' into JSON '"hello"'. If you don't want auto-conversion,
 please provide a JSON
string as event

{
  "ShardId": "shardId-000000000000",
  "SequenceNumber": "49646251411914806775980904340716841045751814812900261890"
}%
```

在命令行中提供一个 JSON 字符串作为事件：

```
$ sam remote invoke KinesisStream --stack-name kinesis-example --event '{"method":
 "GET", "body": ""}'
```

```
Putting record to Kinesis data stream KinesisStream

{
  "ShardId": "shardId-000000000000",
  "SequenceNumber": "49646251411914806775980904492868617924990209230536441858"
}%
```

向 Kinesis Data Streams 应用程序发送一个空事件：

```
$ sam remote invoke KinesisStream --stack-name kinesis-example

Putting record to Kinesis data stream KinesisStream

{
  "ShardId": "shardId-000000000000",
  "SequenceNumber": "49646251411914806775980904866469008589597168190416224258"
}%
```

返回 Amazon SAM CLI JSON 格式的响应：

```
$ sam remote invoke KinesisStream --stack-name kinesis-example --event '{"hello":
 "world"}' --output json

Putting record to Kinesis data stream KinesisStream

{
  "ShardId": "shardId-000000000000",
  "SequenceNumber": "49646251411914806775980905078409420803696667195489648642",
  "ResponseMetadata": {
    "RequestId": "ebbbd307-3e9f-4431-b67c-f0715e9e353e",
    "HTTPStatusCode": 200,
    "HTTPHeaders": {
      "x-amzn-requestid": "ebbbd307-3e9f-4431-b67c-f0715e9e353e",
      "x-amz-id-2": "Q3yBcgTwtPaQTV26IKclbECmZikUYOzKY+CzcxA84ZHgCkc5T2N/
ITWg6RPOQcWw8Gn0tNPcEJBEHyVVqboJAPgCritqsvCu",
      "date": "Thu, 09 Nov 2023 18:13:10 GMT",
      "content-type": "application/x-amz-json-1.1",
      "content-length": "110"
    },
    "RetryAttempts": 0
  }
```

```
}%
```

将 JSON 输出返回到标准输出：

```
$ sam remote invoke KinesisStream --stack-name kinesis-example --event '{"hello":
 "world"}' --output json 1> stdout.log

Putting record to Kinesis data stream KinesisStream


$ cat stdout.log
{
  "ShardId": "shardId-000000000000",
  "SequenceNumber": "49646251411914806775980906397777867595039988349006774274",
  "ResponseMetadata": {
    "RequestId": "f4290006-d84b-b1cd-a9ee-28306eeb2939",
    "HTTPStatusCode": 200,
    "HTTPHeaders": {
      "x-amzn-requestid": "f4290006-d84b-b1cd-a9ee-28306eeb2939",
      "x-amz-id-2": "npCqz
+IBKpoL4sQ1ClbUmxuJlbeA24Fx1UgpIrS6mm2NoIeV2qdZSN5AhNurdssykXajBrXaC9anMhj2eG/h7Hnbf
+bPuotU",
      "date": "Thu, 09 Nov 2023 18:33:26 GMT",
      "content-type": "application/x-amz-json-1.1",
      "content-length": "110"
    },
    "RetryAttempts": 0
  }
}%
```

Lambda 示例

基本示例

通过提供 ARN 作为资源 ID 来调用 Lambda 函数：

```
$ sam remote invoke arn:aws:lambda:us-west-2:012345678910:function:sam-app-
HelloWorldFunction-ohRFEn2RuAvp
```

通过提供逻辑 ID 作为资源 ID 来调用 Lambda 函数：

您还必须使用`--stack-name`选项提供 Amazon CloudFormation 堆栈名称。以下是示例：

```
$ sam remote invoke HelloWorldFunction --stack-name sam-app
```

如果应用程序包含单个 Lambda 函数，则无需指定其逻辑 ID。您只能提供 `--stack-name` 选项。以下是示例：

```
$ sam remote invoke --stack-name sam-app
```

通过提供物理 ID 作为资源 ID 来调用 Lambda 函数：

物理 ID 是在使用进行部署时创建 Amazon CloudFormation的。

```
$ sam remote invoke sam-app-HelloWorldFunction-TZvxQRFNv0k4
```

调用子堆栈的 Lambda 函数：

在本示例中，应用程序包含以下目录结构：

```
lambda-example
### childstack
#   ### function
#   #   ### __init__.py
#   #   ### app.py
#   #   ### requirements.txt
#   ### template.yaml
### events
#   ### event.json
### samconfig.toml
### template.yaml
```

要调用 `childstack` 的 Lambda 函数，我们运行以下命令：

```
$ sam remote invoke ChildStack/HelloWorldFunction --stack-name lambda-example

Invoking Lambda Function HelloWorldFunction

START RequestId: 207a864b-e67c-4307-8478-365b004d4bcd Version: $LATEST
END RequestId: 207a864b-e67c-4307-8478-365b004d4bcd
REPORT RequestId: 207a864b-e67c-4307-8478-365b004d4bcd  Duration: 1.27 ms       Billed
 Duration: 2 ms   Memory Size: 128 MB     Max Memory Used: 36 MB  Init Duration: 111.07
 ms
{"statusCode": 200, "body": "{\"message\": \"Hello\", \"received_event\": {}}"}%
```

调用配置为流式传输响应的 Lambda 函数

在这个例子中，我们使用 Amazon SAM CLI 初始化一个新的无服务器应用程序，该应用程序包含配置为流式传输其响应的 Lambda 函数。我们将应用程序部署到 Amazon Web Services 云 并使用与我们在云中的功能进行交互。`sam remote invoke`

我们首先运行 `sam init` 命令来创建新的无服务器应用程序。我们选择 Lambda 响应流快速入门模板并命名我们的应用程序。lambda-streaming-nodejs-app

```
$ sam init

 You can preselect a particular runtime or package type when using the `sam init`
 experience.
 Call `sam init --help` to learn more.

 Which template source would you like to use?
        1 - AWS Quick Start Templates
        2 - Custom Template Location
 Choice: 1

 Choose an AWS Quick Start application template
        1 - Hello World Example
        ...
        9 - Lambda Response Streaming
        ...
        15 - Machine Learning
 Template: 9

 Which runtime would you like to use?
        1 - go (provided.al2)
        2 - nodejs18.x
        3 - nodejs16.x
 Runtime: 2

 Based on your selections, the only Package type available is Zip.
 We will proceed to selecting the Package type as Zip.

 Based on your selections, the only dependency manager available is npm.
 We will proceed copying the template using npm.

 Would you like to enable X-Ray tracing on the function(s) in your application?  [y/
N]: ENTER
```

```
 Would you like to enable monitoring using CloudWatch Application Insights?
 For more info, please view https://docs.aws.amazon.com/AmazonCloudWatch/latest/
monitoring/cloudwatch-application-insights.html [y/N]: ENTER

 Project name [sam-app]: lambda-streaming-nodejs-app

     -----------------------
     Generating application:
     -----------------------
     Name: lambda-streaming-nodejs-app
     Runtime: nodejs18.x
     Architectures: x86_64
     Dependency Manager: npm
     Application Template: response-streaming
     Output Directory: .
     Configuration file: lambda-streaming-nodejs-app/samconfig.toml

     Next steps can be found in the README file at lambda-streaming-nodejs-app/
README.md


 Commands you can use next
 =========================
 [*] Create pipeline: cd lambda-streaming-nodejs-app && sam pipeline init --bootstrap
 [*] Validate SAM template: cd lambda-streaming-nodejs-app && sam validate
 [*] Test Function in the Cloud: cd lambda-streaming-nodejs-app && sam sync --stack-
name {stack-name} --watch
```

的 Amazon SAM CLI 使用以下结构创建我们的项目：

```
lambda-streaming-nodejs-app
 ### README.md
 ### __tests__
 #   ### unit
 #       ### index.test.js
 ### package.json
 ### samconfig.toml
 ### src
 #   ### index.js
 ### template.yaml
```

以下是 Lambda 函数代码的示例：

```
exports.handler = awslambda.streamifyResponse(
   async (event, responseStream, context) => {
     const httpResponseMetadata = {
       statusCode: 200,
       headers: {
         "Content-Type": "text/html",
         "X-Custom-Header": "Example-Custom-Header"
       }
     };

     responseStream = awslambda.HttpResponseStream.from(responseStream,
 httpResponseMetadata);
     // It's recommended to use a `pipeline` over the `write` method for more complex
 use cases.
     // Learn more: https://docs.aws.amazon.com/lambda/latest/dg/configuration-
response-streaming.html
     responseStream.write("<html>");
     responseStream.write("<p>First write!</p>");

     responseStream.write("<h1>Streaming h1</h1>");
     await new Promise(r => setTimeout(r, 1000));
     responseStream.write("<h2>Streaming h2</h2>");
     await new Promise(r => setTimeout(r, 1000));
     responseStream.write("<h3>Streaming h3</h3>");
     await new Promise(r => setTimeout(r, 1000));

     // Long strings will be streamed
     const loremIpsum1 = "Lorem ipsum dolor sit amet, consectetur adipiscing elit.
 Quisque vitae mi tincidunt tellus ultricies dignissim id et diam. Morbi pharetra eu
 nisi et finibus. Vivamus diam nulla, vulputate et nisl cursus, pellentesque vehicula
 libero. Cras imperdiet lorem ante, non posuere dolor sollicitudin a. Vestibulum ipsum
 lacus, blandit nec augue id, lobortis dictum urna. Vestibulum ante ipsum primis in
 faucibus orci luctus et ultrices posuere cubilia curae; Morbi auctor orci eget tellus
 aliquam, non maximus massa porta. In diam ante, pulvinar aliquam nisl non, elementum
 hendrerit sapien. Vestibulum massa nunc, mattis non congue vitae, placerat in quam.
 Nam vulputate lectus metus, et dignissim erat varius a.";
     responseStream.write(`<p>${loremIpsum1}</p>`);
     await new Promise(r => setTimeout(r, 1000));

     responseStream.write("<p>DONE!</p>");
     responseStream.write("</html>");
     responseStream.end();
   }
```

```
);
```

以下是 template.yaml 文件的示例。Lambda 函数的响应流式传输是使用 FunctionUrlConfig 属性配置的。

```
AWSTemplateFormatVersion: '2010-09-09'
 Transform: AWS::Serverless-2016-10-31

 Description: >
   Sample SAM Template for lambda-streaming-nodejs-app

 Resources:
   StreamingFunction:
     Type: AWS::Serverless::Function
     Properties:
       CodeUri: src/
       Handler: index.handler
       Runtime: nodejs18.x
       Architectures:
         - x86_64
       Timeout: 10
       FunctionUrlConfig:
         AuthType: AWS_IAM
         InvokeMode: RESPONSE_STREAM

 Outputs:
   StreamingFunction:
     Description: "Streaming Lambda Function ARN"
     Value: !GetAtt StreamingFunction.Arn
   StreamingFunctionURL:
     Description: "Streaming Lambda Function URL"
     Value: !GetAtt StreamingFunctionUrl.FunctionUrl
```

通常，您可以使用 sam build 和 sam deploy --guided 来构建和部署生产应用程序。在此示例中，我们将假设开发环境并使用 sam sync 命令来构建和部署应用程序。

> **Note**
>
> 建议在开发环境中使用 sam sync 命令。要了解更多信息，请参阅使用简介 sam sync 同步到 Amazon Web Services 云。

在运行 `sam sync` 之前，我们会验证 `samconfig.toml` 文件中的项目是否配置正确。最重要的是，我们会验证 `stack_name` 和 `watch` 的值。在配置文件中指定了这些值后，我们就不必在命令行中提供它们了。

```
version = 0.1

 [default]
 [default.global.parameters]
 stack_name = "lambda-streaming-nodejs-app"

 [default.build.parameters]
 cached = true
 parallel = true

 [default.validate.parameters]
 lint = true

 [default.deploy.parameters]
 capabilities = "CAPABILITY_IAM"
 confirm_changeset = true
 resolve_s3 = true
 s3_prefix = "lambda-streaming-nodejs-app"
 region = "us-west-2"
 image_repositories = []

 [default.package.parameters]
 resolve_s3 = true

 [default.sync.parameters]
 watch = true

 [default.local_start_api.parameters]
 warm_containers = "EAGER"

 [default.local_start_lambda.parameters]
 warm_containers = "EAGER"
```

接下来，我们运行 `sam sync` 来构建和部署应用程序。由于该`--watch`选项是在我们的配置文件中配置的，因此 Amazon SAM CLI 将构建我们的应用程序、部署我们的应用程序并注意更改。

```
$ sam sync
```

```
 The SAM CLI will use the AWS Lambda, Amazon API Gateway, and AWS StepFunctions APIs to
 upload your code
 without

 performing a CloudFormation deployment. This will cause drift in your CloudFormation
 stack.
 **The sync command should only be used against a development stack**.



 Queued infra sync. Waiting for in progress code syncs to complete...

 Starting infra sync.

 Building codeuri:

 /Users/.../lambda-streaming-nodejs-app/src runtime: nodejs18.x metadata: {}
 architecture: x86_64 functions: StreamingFunction
 package.json file not found. Continuing the build without dependencies.

 Running NodejsNpmBuilder:CopySource


 Build Succeeded

 Successfully packaged artifacts and wrote output template to file /var/
folders/45/5ct135bx3fn2551_ptl5g6_80000gr/T/tmpavrzdhgp.
 Execute the following command to deploy the packaged template
 sam deploy --template-file /var/folders/45/5ct135bx3fn2551_ptl5g6_80000gr/T/
tmpavrzdhgp --stack-name <YOUR STACK NAME>


         Deploying with following values
         ===============================
         Stack name                   : lambda-streaming-nodejs-app
         Region                       : us-west-2
         Disable rollback             : False
         Deployment s3 bucket         : aws-sam-cli-managed-default-samclisam-s3-demo-
bucket-1a4x26zbcdkqr
         Capabilities                 : ["CAPABILITY_NAMED_IAM",
 "CAPABILITY_AUTO_EXPAND"]
         Parameter overrides          : {}
         Signing Profiles             : null
```

```
Initiating deployment
=====================


2023-06-20 12:11:16 - Waiting for stack create/update to complete

CloudFormation events from stack operations (refresh every 0.5 seconds)
-----------------------------------------------------------------------------------------------
ResourceStatus            ResourceType              LogicalResourceId
ResourceStatusReason
-----------------------------------------------------------------------------------------------
CREATE_IN_PROGRESS        AWS::CloudFormation::St   lambda-streaming-
Transformation
                          ack                       nodejs-app
succeeded
CREATE_IN_PROGRESS        AWS::IAM::Role            StreamingFunctionRole     -

CREATE_IN_PROGRESS        AWS::CloudFormation::St   AwsSamAutoDependencyLay   -

                          ack                       erNestedStack

CREATE_IN_PROGRESS        AWS::IAM::Role            StreamingFunctionRole     Resource
creation

Initiated
CREATE_IN_PROGRESS        AWS::CloudFormation::St   AwsSamAutoDependencyLay   Resource
creation
                          ack                       erNestedStack
Initiated
CREATE_COMPLETE           AWS::IAM::Role            StreamingFunctionRole     -

CREATE_COMPLETE           AWS::CloudFormation::St   AwsSamAutoDependencyLay   -

                          ack                       erNestedStack

CREATE_IN_PROGRESS        AWS::Lambda::Function     StreamingFunction         -

CREATE_IN_PROGRESS        AWS::Lambda::Function     StreamingFunction         Resource
creation

Initiated
CREATE_COMPLETE           AWS::Lambda::Function     StreamingFunction         -
```

```
 CREATE_IN_PROGRESS        AWS::Lambda::Url          StreamingFunctionUrl      -

 CREATE_IN_PROGRESS        AWS::Lambda::Url          StreamingFunctionUrl      Resource
 creation

 Initiated
 CREATE_COMPLETE           AWS::Lambda::Url          StreamingFunctionUrl      -

 CREATE_COMPLETE           AWS::CloudFormation::St   lambda-streaming-         -

                           ack                       nodejs-app

 -----------------------------------------------------------------------------------------------

 CloudFormation outputs from deployed stack
 -----------------------------------------------------------------------------------------------
 Outputs

 -----------------------------------------------------------------------------------------------
 Key                 StreamingFunction

 Description         Streaming Lambda Function ARN

 Value               arn:aws:lambda:us-west-2:012345678910:function:lambda-streaming-
nodejs-app-
 StreamingFunction-gUmhO833A0vZ


 Key                 StreamingFunctionURL

 Description         Streaming Lambda Function URL

 Value               https://wxgkcc2dyntgtrwhf2dgdcvylu0rnnof.lambda-url.us-
west-2.on.aws/
 -----------------------------------------------------------------------------------------------



 Stack creation succeeded. Sync infra completed.



 Infra sync completed.
```

现在函数已经部署到云端，我们可以使用 `sam remote invoke` 与函数进行交互。的 Amazon SAM CLI 自动检测到我们的函数已配置为响应流，并立即开始实时输出我们函数的流式响应。

```
$ sam remote invoke StreamingFunction

 Invoking Lambda Function StreamingFunction

 {"statusCode":200,"headers":{"Content-Type":"text/html","X-Custom-Header":"Example-
Custom-Header"}}<html><p>First write!</p><h1>Streaming h1</h1><h2>Streaming h2</
h2><h3>Streaming h3</h3><p>Lorem ipsum dolor sit amet, consectetur adipiscing elit.
 Quisque vitae mi tincidunt tellus ultricies dignissim id et diam. Morbi pharetra eu
 nisi et finibus. Vivamus diam nulla, vulputate et nisl cursus, pellentesque vehicula
 libero. Cras imperdiet lorem ante, non posuere dolor sollicitudin a. Vestibulum ipsum
 lacus, blandit nec augue id, lobortis dictum urna. Vestibulum ante ipsum primis in
 faucibus orci luctus et ultrices posuere cubilia curae; Morbi auctor orci eget tellus
 aliquam, non maximus massa porta. In diam ante, pulvinar aliquam nisl non, elementum
 hendrerit sapien. Vestibulum massa nunc, mattis non congue vitae, placerat in quam.
 Nam vulputate lectus metus, et dignissim erat varius a.</p><p>DONE!</p></html>START
 RequestId: 1e4cdf04-60de-4769-b3a2-c1481982deb4 Version: $LATEST
 END RequestId: 1e4cdf04-60de-4769-b3a2-c1481982deb4
 REPORT RequestId: 1e4cdf04-60de-4769-b3a2-c1481982deb4  Duration: 4088.66 ms
 Billed Duration: 4089 ms        Memory Size: 128 MB     Max Memory Used: 68 MB  Init
 Duration: 168.45 ms
```

当我们修改函数代码时， Amazon SAM CLI 即时检测并立即部署我们的更改。以下是示例 Amazon SAM CLI 对我们的函数代码进行更改后的输出：

```
Syncing Lambda Function StreamingFunction...

 Building codeuri:

 /Users/.../lambda-streaming-nodejs-app/src runtime: nodejs18.x metadata: {}
 architecture:
 x86_64 functions: StreamingFunction

 package.json file not found. Continuing the build without dependencies.

 Running NodejsNpmBuilder:CopySource

 Finished syncing Lambda Function StreamingFunction.

 Syncing Layer StreamingFunctione9cfe924DepLayer...
```

```
SyncFlow [Layer StreamingFunctione9cfe924DepLayer]: Skipping resource update as the

content didn't change

Finished syncing Layer StreamingFunctione9cfe924DepLayer.
```

现在，我们可以再次使用 `sam remote invoke` 与云端函数进行交互并测试我们的更改。

SQS 示例

基本示例

通过提供 ARN 作为资源 ID 来调用 Amazon SQS 队列：

```
$ sam remote invoke arn:aws:sqs:us-west-2:01234567890:sqs-example-4DonhBsjsW1b --
event '{"hello": "world"}' --output json

Sending message to SQS queue MySqsQueue

{
  "MD5OfMessageBody": "49dfdd54b01cbcd2d2ab5e9e5ee6b9b9",
  "MessageId": "4f464cdd-15ef-4b57-bd72-3ad225d80adc",
  "ResponseMetadata": {
    "RequestId": "95d39377-8323-5ef0-9223-ceb198bd09bd",
    "HTTPStatusCode": 200,
    "HTTPHeaders": {
      "x-amzn-requestid": "95d39377-8323-5ef0-9223-ceb198bd09bd",
      "date": "Wed, 08 Nov 2023 23:27:26 GMT",
      "content-type": "application/x-amz-json-1.0",
      "content-length": "106",
      "connection": "keep-alive"
    },
    "RetryAttempts": 0
  }
}%
```

Step Functions 示例

基本示例

通过提供物理 ID 作为资源 ID 来调用状态机：

首先，我们使用 `sam list resources` 来获取物理 ID：

```
$ sam list resources --stack-name state-machine-example --output json

[
  {
    "LogicalResourceId": "HelloWorldStateMachine",
    "PhysicalResourceId": "arn:aws:states:us-
west-2:513423067560:stateMachine:HelloWorldStateMachine-z69tFEUx0F66"
  },
  {
    "LogicalResourceId": "HelloWorldStateMachineRole",
    "PhysicalResourceId": "simple-state-machine-HelloWorldStateMachineRole-
PduA0BDGuFXw"
  }
]
```

接下来，我们使用物理 ID 作为资源 ID 来调用状态机。我们通过 `--event` 选项在命令行传递事件：

```
$ sam remote invoke arn:aws:states:us-
west-2:01234567890:stateMachine:HelloWorldStateMachine-z69tFEUx0F66 --
event '{"is_developer": true}'

Invoking Step Function arn:aws:states:us-
west-2:01234567890:stateMachine:HelloWorldStateMachine-z69tFEUx0F66
"Hello Developer World"%
```

通过传递一个空事件来调用状态机：

```
$ sam remote invoke HelloWorldStateMachine --stack-name state-machine-example

Invoking Step Function HelloWorldStateMachine

"Hello World"%
```

相关链接

有关`sam remote invoke`和使用的文档 Amazon SAM CLI，请参阅以下内容：

- sam remote invoke
- Amazon SAM CLI 故障排除

使用自动执行本地集成测试 Amazon SAM

虽然您可以使用使用测试简介 sam local invoke手动测试代码，但 Amazon SAM 也可以使用自动集成测试来测试代码。集成测试可帮助您在开发周期的早期发现问题，提高代码质量，节省时间，同时降低成本。

要在中创作自动集成测试 Amazon SAM，您需要先对本地 Lambda 函数运行测试，然后再部署到云端。 Amazon 使用测试简介 sam local start-lambda 命令启动模拟 Lambda 调用端点的本地端点。您可以从自动测试中调用它。由于此端点模拟 Lambda 调用端点，因此您可以编写一次测试并针对本地 Lambda 函数或部署的 Lambda 函数运行测试（无需做任何修改）。您还可以针对您的 CI/CD 管道中部署的 Amazon SAM 堆栈运行相同的测试。

以下是此过程的工作原理：

1. 启动本地 Lambda 端点。

   在包含您的 Amazon SAM 模板的目录中运行以下命令来启动本地 Lambda 终端节点：

   ```
   sam local start-lambda
   ```

   此命令在 `http://127.0.0.1:3001` 处启动模拟 Amazon Lambda的本地端点。您可以针对此本地 Lambda 端点运行自动测试。当您使用 Amazon CLI 或 SDK 调用此终端节点时，它会在本地执行请求中指定的 Lambda 函数并返回响应。

2. 针对本地 Lambda 端点运行集成测试。

   在集成测试中，您可以使用 Amazon 软件开发工具包调用带有测试数据的 Lambda 函数，等待响应，然后验证响应是否符合您的预期。要本地运行集成测试，您应配置 Amazon 软件开发工具包以发送 Lambda Invoke API 调用，以调用上一步中启动的本地 Lambda 端点。

   以下是 Python 示例（其他语言 Amazon SDKs 的配置类似）：

   ```
   import boto3
   import botocore

   # Set "running_locally" flag if you are running the integration test locally
   running_locally = True

   if running_locally:

       # Create Lambda SDK client to connect to appropriate Lambda endpoint
   ```

```
    lambda_client = boto3.client('lambda',
        region_name="us-west-2",
        endpoint_url="http://127.0.0.1:3001",
        use_ssl=False,
        verify=False,
        config=botocore.client.Config(
            signature_version=botocore.UNSIGNED,
            read_timeout=15,
            retries={'max_attempts': 0},
        )
    )
else:
    lambda_client = boto3.client('lambda')


# Invoke your Lambda function as you normally usually do. The function will run
# locally if it is configured to do so
response = lambda_client.invoke(FunctionName="HelloWorldFunction")

# Verify the response
assert response == "Hello World"
```

您可以将 `running_locally` 设置为 `False`，使用此代码来测试已部署的 Lambda 函数。这将设置要在 Amazon 云端连接的 Amazon SDK。 Amazon Lambda

使用生成示例事件有效负载 Amazon SAM

要测试您的 Lambda 函数，您可以生成和自定义示例事件负载，这些负载会模仿您的 Lambda 函数在其他服务触发时将收到的数据。 Amazon 其中包括 API Gateway、 Amazon CloudFormation、Amazon S3 等服务。

生成示例事件有效载荷可帮助您使用各种不同的输入来测试 Lambda 函数的行为，而无需在实时环境中工作。与手动创建 Amazon 服务事件样本来测试函数相比，这种方法还可以节省时间。

要查看可以为其生成示例事件负载的服务的完整列表，请使用以下命令：

```
sam local generate-event --help
```

要查看可用于特定服务的选项列表，请使用以下命令：

```
sam local generate-event [SERVICE] --help
```

示例：

```
#Generates the event from S3 when a new object is created
sam local generate-event s3 put

# Generates the event from S3 when an object is deleted
sam local generate-event s3 delete
```

使用调试您的无服务器应用程序 Amazon SAM

测试应用程序后，您就可以调试发现的任何问题了。使用 Amazon SAM 命令行界面 (CLI)，您可以先在本地测试和调试您的无服务器应用程序，然后再将其上传到 Amazon 云端。调试应用程序可识别并修复应用程序中的问题或错误。

您可以使用 Amazon SAM 执行分步调试，这是一种逐行运行代码或指令的方法。当您在调试模式下本地调用 Lambda 函数时 Amazon SAM CLI，然后可以为其附加调试器。借助调试程序，您可以逐行分步调试代码，查看不同变量的值，并像处理任何其他应用程序一样修复问题。在完成打包和部署应用程序的步骤之前，您可以验证应用程序是否按预期运行，调试问题，并修正任何问题。

> **Note**
>
> 如果应用程序包含一个或多个层，则在本地运行和调试应用程序时，层包会下载并缓存在本地主机上。有关更多信息，请参阅 如何在本地缓存层。

主题

- 使用本地调试函数 Amazon SAM
- 使用调试时传递多个运行时参数 Amazon SAM
- 使用 Amazon CloudFormation Linter 验证您的 Amazon SAM 应用程序

使用本地调试函数 Amazon SAM

您可以 Amazon SAM 与各种 Amazon 工具包和调试器配合使用，在本地测试和调试您的无服务器应用程序。通过逐步调试 Lambda 函数，您可以在本地环境中逐行或逐条指令地识别和修复应用程序中的问题。

您可以执行本地逐步调试的一些方法包括设置断点、检查变量以及逐行执行函数代码。本地分步调试使您可以发现和解决可能在云中遇到的问题，从而缩短反馈循环。

您可以使用 Amazon 工具包进行调试，也可以在调试模式 Amazon SAM 下运行。有关详细信息，请参阅本节中的主题。

使用 Amazon 工具包

Amazon 工具包是集成开发环境 (IDE) 插件，可让您执行许多常见的调试任务，例如设置断点、检查变量和逐行执行函数代码。 Amazon 工具包使您可以更轻松地开发、调试和部署使用构建的无服务器应用程序。 Amazon SAM它们提供了构建、测试、调试、部署和调用集成到 IDE 中的 Lambda 函数的体验。

有关可与之配合使用的 Amazon 工具包的更多信息 Amazon SAM，请参阅以下内容：

- Amazon Toolkit for Visual Studio Code
- Amazon Cloud9
- Amazon Toolkit for JetBrains

有各种各样的 Amazon 工具包适用于 IDEs 和运行时的不同组合。下表列出了支持应用程序分步调试的常用 IDE/Runtime 组合： Amazon SAM

IDE	运行时	Amazon 工具包	逐步调试说明
Visual Studio Code	• Node.js • Python • .NET • Java • Go	Amazon Toolkit for Visual Studio Code	《Amazon Toolkit for Visual Studio Code 用户指南》中的使用 Amazon Serverless Application
Amazon Cloud9	• Node.js • Python	Amazon Cloud9，启用 Amazon 工具包后 [1]	使用《Amazon Cloud9 用户指南》中的 Amazon Toolkit 使用 Amazon 无服务器应用程序。
WebStorm	Node.js	Amazon Toolkit for JetBrains[2]	在 Amazon Toolkit for JetBrains 中运行（调用）或调试本地函数
PyCharm	Python	Amazon Toolkit for JetBrains[2]	在 Amazon Toolkit for JetBrains 中运行（调用）或调试本地函数

IDE	运行时	Amazon 工具包	逐步调试说明
Rider	.NET	Amazon Toolkit for JetBrains[2]	在 Amazon Toolkit for JetBrains 中运行（调用）或调试本地函数
IntelliJ	Java	Amazon Toolkit for JetBrains[2]	在 Amazon Toolkit for JetBrains 中运行（调用）或调试本地函数
GoLand	Go	Amazon Toolkit for JetBrains[2]	在 Amazon Toolkit for JetBrains 中运行（调用）或调试本地函数

备注：

1. Amazon Cloud9 要使用逐步调试 Amazon SAM 应用程序，必须启用 Amazon Toolkit。有关更多信息，请参阅《Amazon Cloud9 用户指南》中的 “启用 Amazon 工具包”。
2. 要使用分步调试 Amazon SAM 应用程序，必须先按照中安装中的说明进行安装和配置。 Amazon Toolkit for JetBrains Amazon Toolkit for JetBrainsAmazon Toolkit for JetBrains

在调试模式下在 Amazon SAM 本地运行

除了与 Amazon Toolkits 集成外，您还可以 Amazon SAM 在 “调试模式” 下运行以连接到第三方调试器，例如 pt vsd 或 delve。

要 Amazon SAM 在调试模式下运行，请使用命令sam local invoke或sam local start-api使用`--debug-port`或`-d`选项。

例如：

```
# Invoke a function locally in debug mode on port 5858
sam local invoke -d 5858 <function logical id>

# Start local API Gateway in debug mode on port 5858
sam local start-api -d 5858
```

> **Note**
>
> 如果使用 `sam local start-api`，则本地 API Gateway 实例会公开您的所有 Lambda 函数。但是，因为您能指定一个调试端口，所以每次只能调试一个函数。你需要先调用你的 API Amazon SAM CLI 绑定到允许调试器连接的端口。

使用调试时传递多个运行时参数 Amazon SAM

您可以选择传递额外的运行时参数， Amazon SAM 以更有效地检查问题和解决变量。这样做可以增强调试过程的可控性和灵活性，从而帮助您自定义运行时配置和环境。

要在调试函数时传递其他运行时参数，请使用环境变量 `DEBUGGER_ARGS`。这会将一串参数直接传递到 run 命令中 Amazon SAM CLI 用于启动您的函数。

例如，如果你想在 Python 函数的运行时加载像我KPdb 这样的调试器，你可以将以下内容传递为`DEBUGGER_ARGS: -m ikpdb --ikpdb-port=5858 --ikpdb-working-directory=/var/task/ --ikpdb-client-working-directory=/myApp --ikpdb-address=0.0.0.0`。这将在运行KPdb 时加载你指定的其他参数。

在这种情况下，你的已满 Amazon SAM CLI 命令将是：

```
DEBUGGER_ARGS="-m ikpdb --ikpdb-port=5858 --ikpdb-working-directory=/var/task/ --ikpdb-
client-working-directory=/myApp --ikpdb-address=0.0.0.0" echo {} | sam local invoke -d
 5858 myFunction
```

您可以将调试器参数传递给所有运行时的函数。

使用 Amazon CloudFormation Linter 验证您的 Amazon SAM 应用程序

Amazon CloudFormation Linter（cfn-lint）是一个开源工具，可用于对模板进行详细验证。 Amazon CloudFormation CFN-Lint 包含以资源规范为指导的 Amazon CloudFormation 规则。使用 cfn-lint 将您的资源与这些规则进行比较，以接收有关错误、警告或信息建议的详细消息。或者，创建自己的自定义规则进行验证。要了解有关 cfn-lint 的更多信息，请参阅存储库中的 cfn- lint。Amazon CloudFormation GitHub

你可以使用 cfn-lint 通过 Amazon SAM 命令行界面验证你的 Amazon Serverless Application Model (Amazon SAM) 模板 (Amazon SAM CLI) 通过sam validate使用--lint选项运行。

```
sam validate --lint
```

要自定义 cfn-lint 行为，例如创建自定义规则或指定验证选项，可以定义配置文件。要了解更多信息，请参阅 cf n-lint 存储库中的 Conf Amazon CloudFormation GitHub ig 文件。运行 sam validate --lint 时，将应用配置文件中定义的 cfn-lint 行为。

示例

对模板执行 cfn-lint 验证 Amazon SAM

```
sam validate --lint --template myTemplate.yaml
```

了解更多

要了解有关 sam validate 命令的更多信息，请参阅 sam validate。

使用部署您的应用程序和资源 Amazon SAM

部署应用程序会预置和配置 Amazon 云中的 Amazon 资源，从而使您的应用程序在云中运行。Amazon SAM Amazon CloudFormation用作其底层部署机制。 Amazon SAM 使用您在运行sam build命令时创建的构建工件作为部署无服务器应用程序的标准输入。

使用 Amazon SAM，您可以手动部署无服务器应用程序，也可以自动部署。要实现部署自动化，您可以使用带有您选择的持续集成和持续部署 (CI/CD) 系统的 Amazon SAM 管道。您的部署管线是一系列自动执行的步骤，用于发布无服务器应用程序的新版本。

本节中的主题提供了有关自动部署和手动部署的指导。要手动部署应用程序，请使用 Amazon SAM CLI 命令。要自动执行部署，请参阅本节中的主题。这些主题特别提供了有关使用管线和 CI/CD 系统自动部署的深入内容。其中包括生成入门管线、设置自动化、排查部署故障、使用 OpenID Connect（OIDC）用户身份验证以及在部署时上传本地文件。

主题

- 使用部署简介 Amazon SAM
- 用于部署应用程序的选项 Amazon SAM
- 使用 CI/CD 系统和管道进行部署 Amazon SAM
- 使用简介 sam sync 同步到 Amazon Web Services 云

使用部署简介 Amazon SAM

使用 Amazon Serverless Application Model 命令行界面 (Amazon SAM CLI) `sam deploy` 命令将您的无服务器应用程序部署到。 Amazon Web Services 云

- 有关简介 Amazon SAM CLI，请参阅 那是什么 Amazon SAMCLI？。
- 有关 `sam deploy` 命令选项的列表，请参阅 sam deploy。
- 有关在典型开发工作流程中使用 `sam deploy` 的示例，请参阅步骤 3：将您的应用程序部署到 Amazon Web Services 云。

主题

- 先决条件
- 使用 sam deploy 部署应用程序

- 最佳实践
- sam deploy 的选项
- 故障排除
- 示例
- 了解更多

先决条件

要使用`sam deploy`，请安装 Amazon SAM CLI 通过完成以下操作：

- Amazon SAM 先决条件.
- 安装 Amazon SAM CLI.

我们建议您在使用 `sam deploy` 之前初步了解以下主题：

- 正在配置 Amazon SAM CLI.
- 在中创建您的应用程序 Amazon SAM.
- 建筑简介 Amazon SAM.

使用 sam deploy 部署应用程序

首次部署无服务器应用程序时，请使用 `--guided` 选项。的 Amazon SAM CLI 将引导您完成交互式流程，以配置应用程序的部署设置。

使用交互式流程部署应用程序

1. 转到项目的根目录。此位置与您的 Amazon SAM 模板相同。

```
$ cd sam-app
```

2. 运行以下命令：

```
$ sam deploy --guided
```

3. 在互动流程中， Amazon SAM CLI 提示您配置应用程序部署设置的选项。

   方括号 (`[ ]`) 表示默认值。将答案留空以选择默认值。可从以下配置文件获得默认值：

- ~/.aws/config— 您的常规 Amazon 账户设置。
- ~/.aws/credentials— 您的 Amazon 账户凭证。
- *<project>*/samconfig.toml - 项目的配置文件。

通过回答 “来提供值” Amazon SAM CLI 提示。例如，可以输入 **y**（表示是）、**n**（表示否）或字符串值。

的 Amazon SAM CLI 将你的回复写到你的项目samconfig.toml文件中。对于后续部署，您可以使用 sam deploy 和这些配置的值进行部署。要重新配置这些值，请再次使用 sam deploy --guided 或直接修改配置文件。

下面是一个示例输出：

```
sam-app $ sam deploy --guided

Configuring SAM deploy
======================

        Looking for config file [samconfig.toml] :  Found
        Reading default arguments  :  Success

        Setting default arguments for 'sam deploy'
        =========================================
        Stack Name [sam-app]: ENTER
        AWS Region [us-west-2]: ENTER
        #Shows you resources changes to be deployed and require a 'Y' to initiate
 deploy
        Confirm changes before deploy [Y/n]: ENTER
        #SAM needs permission to be able to create roles to connect to the
 resources in your template
        Allow SAM CLI IAM role creation [Y/n]: ENTER
        #Preserves the state of previously provisioned resources when an operation
 fails
        Disable rollback [y/N]: ENTER
        HelloWorldFunction may not have authorization defined, Is this okay? [y/
N]: y
        Save arguments to configuration file [Y/n]: ENTER
        SAM configuration file [samconfig.toml]: ENTER
        SAM configuration environment [default]: ENTER
```

4. 接下来， Amazon SAM CLI 将您的应用程序部署到。 Amazon Web Services 云在部署过程中，命令提示符中会显示进度。以下是部署的主要阶段：

   - 对于 Amazon Lambda 功能打包为.zip 文件存档的应用程序， Amazon SAM CLI 将包裹压缩并上传到亚马逊简单存储服务 (Amazon S3) 存储桶。如有必要， Amazon SAM CLI 将创建一个新的存储桶。
   - 对于将 Lambda 函数打包为容器映像的应用程序， Amazon SAM CLI 将图片上传到亚马逊 Elastic Container Registry (Amazon ECR)。如有必要， Amazon SAM CLI 将创建一个新的存储库。
   - 的 Amazon SAM CLI 创建 Amazon CloudFormation 更改集并将您的应用程序 Amazon CloudFormation 作为堆栈部署到其中。
   - 的 Amazon SAM CLI 使用您的 Lambda 函数的新`CodeUri`值修改已部署的 Amazon SAM 模板。

   以下是示例 Amazon SAM CLI 部署输出：

```
        Looking for resources needed for deployment:

        Managed S3 bucket: aws-sam-cli-managed-default-samclisam-s3-demo-
bucket-1a4x26zbcdkqr
        A different default S3 bucket can be set in samconfig.toml and auto
 resolution of buckets turned off by setting resolve_s3=False

        Parameter "stack_name=sam-app" in [default.deploy.parameters] is defined as
 a global parameter [default.global.parameters].
        This parameter will be only saved under [default.global.parameters] in /
Users/.../sam-app/samconfig.toml.

        Saved arguments to config file
        Running 'sam deploy' for future deployments will use the parameters saved
 above.
        The above parameters can be changed by modifying samconfig.toml
        Learn more about samconfig.toml syntax at
        https://docs.aws.amazon.com/serverless-application-model/latest/
developerguide/serverless-sam-cli-config.html

        Uploading to sam-app-zip/da3c598813f1c2151579b73ad788cac8  262144 / 619839
 (42.29%)Uploading to sam-app-zip/da3c598813f1c2151579b73ad788cac8  524288 / 619839
  (84.58%)Uploading to sam-app-zip/da3c598813f1c2151579b73ad788cac8  619839 /
 619839  (100.00%)
```

```
        Deploying with following values
        ===============================
        Stack name                   : sam-app
        Region                       : us-west-2
        Confirm changeset            : True
        Disable rollback             : False
        Deployment s3 bucket         : aws-sam-cli-managed-default-samclisam-s3-
demo-bucket-1a4x26zbcdkqr
        Capabilities                 : ["CAPABILITY_IAM"]
        Parameter overrides          : {}
        Signing Profiles             : {}

Initiating deployment
=====================

        Uploading to sam-app-zip/be84c20f868068e4dc4a2c11966edf2d.template  1212 /
 1212  (100.00%)


Waiting for changeset to be created..

CloudFormation stack changeset
-----------------------------------------------------------------------------------------------
Operation                LogicalResourceId        ResourceType
 Replacement
-----------------------------------------------------------------------------------------------
+ Add                    HelloWorldFunctionHell   AWS::Lambda::Permissio   N/A

                         oWorldPermissionProd     n

+ Add                    HelloWorldFunctionRole   AWS::IAM::Role           N/A

+ Add                    HelloWorldFunction       AWS::Lambda::Function    N/A

+ Add                    ServerlessRestApiDeplo   AWS::ApiGateway::Deplo   N/A

                         yment47fc2d5f9d          yment

+ Add                    ServerlessRestApiProdS   AWS::ApiGateway::Stage   N/A

                         tage
```

```
+ Add                        ServerlessRestApi        AWS::ApiGateway::RestA   N/A

                                                      pi

-----------------------------------------------------------------------------------------------------


Changeset created successfully. arn:aws:cloudformation:us-
west-2:012345678910:changeSet/samcli-deploy1680559234/d9f58a77-98bc-41cd-
b9f4-433a5a450d7a


Previewing CloudFormation changeset before deployment
======================================================
Deploy this changeset? [y/N]: y

2023-04-03 12:00:50 - Waiting for stack create/update to complete

CloudFormation events from stack operations (refresh every 5.0 seconds)
-----------------------------------------------------------------------------------------------------
ResourceStatus           ResourceType             LogicalResourceId
 ResourceStatusReason
-----------------------------------------------------------------------------------------------------
CREATE_IN_PROGRESS       AWS::IAM::Role           HelloWorldFunctionRole   -

CREATE_IN_PROGRESS       AWS::IAM::Role           HelloWorldFunctionRole   Resource
 creation

 Initiated
CREATE_COMPLETE          AWS::IAM::Role           HelloWorldFunctionRole   -

CREATE_IN_PROGRESS       AWS::Lambda::Function    HelloWorldFunction       -

CREATE_IN_PROGRESS       AWS::Lambda::Function    HelloWorldFunction       Resource
 creation

 Initiated
CREATE_COMPLETE          AWS::Lambda::Function    HelloWorldFunction       -

CREATE_IN_PROGRESS       AWS::ApiGateway::RestA   ServerlessRestApi        -

                         pi

```

```
CREATE_IN_PROGRESS       AWS::ApiGateway::RestA   ServerlessRestApi        Resource
 creation
                         pi
 Initiated
CREATE_COMPLETE          AWS::ApiGateway::RestA   ServerlessRestApi        -

                         pi

CREATE_IN_PROGRESS       AWS::Lambda::Permissio   HelloWorldFunctionHell   -

                         n                        oWorldPermissionProd

CREATE_IN_PROGRESS       AWS::ApiGateway::Deplo   ServerlessRestApiDeplo   -

                         yment                    yment47fc2d5f9d

CREATE_IN_PROGRESS       AWS::Lambda::Permissio   HelloWorldFunctionHell   Resource
 creation
                         n                        oWorldPermissionProd
 Initiated
CREATE_IN_PROGRESS       AWS::ApiGateway::Deplo   ServerlessRestApiDeplo   Resource
 creation
                         yment                    yment47fc2d5f9d
 Initiated
CREATE_COMPLETE          AWS::ApiGateway::Deplo   ServerlessRestApiDeplo   -

                         yment                    yment47fc2d5f9d

CREATE_IN_PROGRESS       AWS::ApiGateway::Stage   ServerlessRestApiProdS   -

                                                  tage

CREATE_IN_PROGRESS       AWS::ApiGateway::Stage   ServerlessRestApiProdS   Resource
 creation
                                                  tage
 Initiated
CREATE_COMPLETE          AWS::ApiGateway::Stage   ServerlessRestApiProdS   -

                                                  tage

CREATE_COMPLETE          AWS::Lambda::Permissio   HelloWorldFunctionHell   -

                         n                        oWorldPermissionProd
```

```
CREATE_COMPLETE          AWS::CloudFormation::S   sam-app-zip              -

                         tack

-----------------------------------------------------------------------------------------------

CloudFormation outputs from deployed stack
-----------------------------------------------------------------------------------------------
Outputs

-----------------------------------------------------------------------------------------------
Key                 HelloWorldFunctionIamRole

Description         Implicit IAM Role created for Hello World function

Value               arn:aws:iam::012345678910:role/sam-app-zip-

HelloWorldFunctionRole-11ZOGSCG28H0M


Key                 HelloWorldApi

Description         API Gateway endpoint URL for Prod stage for Hello World
 function
Value               https://njzfhdmls0.execute-api.us-west-2.amazonaws.com/Prod/
hello/

Key                 HelloWorldFunction

Description         Hello World Lambda Function ARN

Value               arn:aws:lambda:us-west-2:012345678910:function:sam-app-

HelloWorldFunction-XPqNX4TBu7qn

-----------------------------------------------------------------------------------------------



Successfully created/updated stack - sam-app-zip in us-west-2
```

5. 要查看您部署的应用程序，请执行以下操作：

   1. 使用 https://console.aws.amazon.com/cloudformation 网址直接打开 Amazon CloudFormation 控制台。

2. 选择堆栈。
3. 通过应用程序名称识别堆栈并选择它。

在部署之前验证更改

你可以配置 Amazon SAM CLI 以显示您的 Amazon CloudFormation 更改集并在部署之前请求确认。

在部署之前确认更改

1. 使用 `sam deploy --guided` 进行部署时，在部署之前输入 **Y** 确认更改。

```
#Shows you resources changes to be deployed and require a 'Y' to initiate deploy
Confirm changes before deploy [Y/n]: Y
```

   或者，可以使用以下方法修改 `samconfig.toml` 文件：

```
[default.deploy]
[default.deploy.parameters]
confirm_changeset = true
```

2. 在部署期间， Amazon SAM CLI 部署前会要求您确认更改。以下是示例：

```
Waiting for changeset to be created..

CloudFormation stack changeset
-------------------------------------------------------------------------------------------
Operation                LogicalResourceId        ResourceType
 Replacement
-------------------------------------------------------------------------------------------
+ Add                    HelloWorldFunctionHell   AWS::Lambda::Permissio   N/A

                         oWorldPermissionProd     n

+ Add                    HelloWorldFunctionRole   AWS::IAM::Role           N/A

+ Add                    HelloWorldFunction       AWS::Lambda::Function    N/A

+ Add                    ServerlessRestApiDeplo   AWS::ApiGateway::Deplo   N/A

                         yment47fc2d5f9d          yment
```

```
+ Add                     ServerlessRestApiProdS   AWS::ApiGateway::Stage   N/A

                          tage

+ Add                     ServerlessRestApi        AWS::ApiGateway::RestA   N/A

                                                   pi

-----------------------------------------------------------------------------------------------

Changeset created successfully. arn:aws:cloudformation:us-
west-2:012345678910:changeSet/samcli-deploy1680559234/d9f58a77-98bc-41cd-
b9f4-433a5a450d7a
Previewing CloudFormation changeset before deployment
=====================================================
Deploy this changeset? [y/N]: y
```

在部署过程中指定其他参数

您可以在部署时指定要配置的其他参数值。若要这样做，只需在部署过程中修改 Amazon SAM 模板并配置参数值。

指定其他参数

1. 修改 Amazon SAM 模板的`Parameters`部分。以下是示例：

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
...
Globals:
...
Parameters:
  DomainName:
    Type: String
    Default: example
    Description: Domain name
```

2. 运行 `sam deploy --guided`。下面是一个示例输出：

```
sam-app $ sam deploy --guided

Configuring SAM deploy
```

```
=======================

        Looking for config file [samconfig.toml] :  Found
        Reading default arguments  :  Success

        Setting default arguments for 'sam deploy'
        =========================================
        Stack Name [sam-app-zip]: ENTER
        AWS Region [us-west-2]: ENTER
        Parameter DomainName [example]: ENTER
```

配置 Lambda 函数的代码签名

您可以在部署时配置 Lambda 函数的代码签名。为此，您可以修改 Amazon SAM 模板并在部署期间配置代码签名。

配置代码签名

1. 在 Amazon SAM 模板`CodeSigningConfigArn`中指定。以下是示例：

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
...
Resources:
  HelloWorldFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: hello_world/
      Handler: app.lambda_handler
      Runtime: python3.7
      CodeSigningConfigArn: arn:aws:lambda:us-east-1:111122223333:code-signing-
config:csc-12e12345db1234567
```

2. 运行 `sam deploy --guided`。的 Amazon SAM CLI 将提示您配置代码签名。下面是一个示例输出：

```
#Found code signing configurations in your function definitions
Do you want to sign your code? [Y/n]: ENTER
#Please provide signing profile details for the following functions & layers
#Signing profile details for function 'HelloWorld'
Signing Profile Name:
Signing Profile Owner Account ID (optional):
```

```
#Signing profile details for layer 'MyLayer', which is used by functions
 {'HelloWorld'}
Signing Profile Name:
Signing Profile Owner Account ID (optional):
```

最佳实践

- 使用时`sam deploy`，Amazon SAM CLI 部署位于`.aws-sam`目录中的应用程序的构建工件。如果您更改了应用程序的原始文件，请在部署之前运行 `sam build` 以更新 `.aws-sam` 目录。
- 首次部署应用程序时，使用 `sam deploy --guided` 来配置部署设置。对于后续部署，您可以使用 `sam deploy` 和配置的设置进行部署。

sam deploy 的选项

以下是 `sam deploy` 的常用选项：有关全部选项的列表，请参阅 sam deploy。

使用有指导的交互式流程部署应用程序

使用 `--guided` 选项通过交互式流程配置应用程序的部署设置。以下是示例：

```
$ sam deploy --guided
```

应用程序的部署设置保存在项目的 `samconfig.toml` 文件中。要了解更多信息，请参阅配置项目设置。

故障排除

要排除故障 Amazon SAM CLI，请参阅 Amazon SAM CLI 故障排除。

示例

部署包含打包为 .zip 文件存档的 Lambda 函数的 Hello World 应用程序

有关示例，请参阅 Hello World 应用程序教程中的步骤 3：将您的应用程序部署到 Amazon Web Services 云。

部署包含打包为容器映像的 Lambda 函数的 Hello World 应用程序

首先，使用 `sam init` 创建 Hello World 应用程序。在交互式流程中，选择 `Python3.9` 运行时和 `Image` 包类型。

```
$ sam init
...
Which template source would you like to use?
        1 - AWS Quick Start Templates
        2 - Custom Template Location
Choice: 1

Choose an AWS Quick Start application template
        1 - Hello World Example
        2 - Multi-step workflow
        ...
Template: 1

Use the most popular runtime and package type? (Python and zip) [y/N]:  ENTER

Which runtime would you like to use?
        1 - aot.dotnet7 (provided.al2)
        ...
        15 - nodejs12.x
        16 - python3.9
        17 - python3.8
        ...
Runtime: 16

What package type would you like to use?
        1 - Zip
        2 - Image
Package type: 2

Based on your selections, the only dependency manager available is pip.
We will proceed copying the template using pip.
...
Project name [sam-app]: ENTER

    -----------------------
    Generating application:
    -----------------------
    Name: sam-app
```

```
    Base Image: amazon/python3.9-base
    Architectures: x86_64
    Dependency Manager: pip
    Output Directory: .
    Configuration file: sam-app/samconfig.toml

    Next steps can be found in the README file at sam-app/README.md
...
```

然后，运行 cd 以进入项目的根目录并运行 sam build。的 Amazon SAM CLI 使用在本地构建我们的 Lambda 函数 Docker.

```
sam-app $ sam build
Building codeuri: /Users/.../sam-app runtime: None metadata: {'Dockerfile':
 'Dockerfile', 'DockerContext': '/Users/.../sam-app/hello_world', 'DockerTag':
 'python3.9-v1'} architecture: x86_64 functions: HelloWorldFunction
Building image for HelloWorldFunction function
Setting DockerBuildArgs: {} for HelloWorldFunction function
Step 1/5 : FROM public.ecr.aws/lambda/python:3.9
 ---> 0a5e3da309aa
Step 2/5 : COPY requirements.txt ./
 ---> abc4e82e85f9
Step 3/5 : RUN python3.9 -m pip install -r requirements.txt -t .
 ---> [Warning] The requested image's platform (linux/amd64) does not match the
 detected host platform (linux/arm64/v8) and no specific platform was requested
 ---> Running in 43845e7aa22d
Collecting requests
  Downloading requests-2.28.2-py3-none-any.whl (62 kB)
     ######################################## 62.8/62.8 KB 829.5 kB/s eta 0:00:00
Collecting idna<4,>=2.5
  Downloading idna-3.4-py3-none-any.whl (61 kB)
     ######################################## 61.5/61.5 KB 2.4 MB/s eta 0:00:00
Collecting charset-normalizer<4,>=2
  Downloading charset_normalizer-3.1.0-cp39-cp39-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl (199 kB)
     ######################################## 199.2/199.2 KB 2.1 MB/s eta 0:00:00
Collecting certifi>=2017.4.17
  Downloading certifi-2022.12.7-py3-none-any.whl (155 kB)
     ######################################## 155.3/155.3 KB 10.2 MB/s eta 0:00:00
Collecting urllib3<1.27,>=1.21.1
  Downloading urllib3-1.26.15-py2.py3-none-any.whl (140 kB)
     ######################################## 140.9/140.9 KB 9.1 MB/s eta 0:00:00
Installing collected packages: urllib3, idna, charset-normalizer, certifi, requests
```

```
Successfully installed certifi-2022.12.7 charset-normalizer-3.1.0 idna-3.4
 requests-2.28.2 urllib3-1.26.15
Removing intermediate container 43845e7aa22d
 ---> cab8ace899ce
Step 4/5 : COPY app.py ./
 ---> 4146f3cd69f2
Step 5/5 : CMD ["app.lambda_handler"]
 ---> [Warning] The requested image's platform (linux/amd64) does not match the
 detected host platform (linux/arm64/v8) and no specific platform was requested
 ---> Running in f4131ddffb31
Removing intermediate container f4131ddffb31
 ---> d2f5180b2154
Successfully built d2f5180b2154
Successfully tagged helloworldfunction:python3.9-v1


Build Succeeded

Built Artifacts  : .aws-sam/build
Built Template   : .aws-sam/build/template.yaml

Commands you can use next
=========================
[*] Validate SAM template: sam validate
[*] Invoke Function: sam local invoke
[*] Test Function in the Cloud: sam sync --stack-name {{stack-name}} --watch
[*] Deploy: sam deploy --guided
```

然后，运行 `sam deploy --guided` 部署应用程序。的 Amazon SAM CLI 指导我们配置部署设置。然后， Amazon SAM CLI 将我们的应用程序部署到。 Amazon Web Services 云

```
sam-app $ sam deploy --guided

Configuring SAM deploy
======================

        Looking for config file [samconfig.toml] :  Found
        Reading default arguments  :  Success

        Setting default arguments for 'sam deploy'
        =========================================
        Stack Name [sam-app]: ENTER
        AWS Region [us-west-2]: ENTER
```

```
        #Shows you resources changes to be deployed and require a 'Y' to initiate
 deploy
        Confirm changes before deploy [Y/n]: ENTER
        #SAM needs permission to be able to create roles to connect to the resources in
 your template
        Allow SAM CLI IAM role creation [Y/n]: ENTER
        #Preserves the state of previously provisioned resources when an operation
 fails
        Disable rollback [y/N]: ENTER
        HelloWorldFunction may not have authorization defined, Is this okay? [y/N]: y
        Save arguments to configuration file [Y/n]: ENTER
        SAM configuration file [samconfig.toml]: ENTER
        SAM configuration environment [default]: ENTER

        Looking for resources needed for deployment:

        Managed S3 bucket: aws-sam-cli-managed-default-samclisam-s3-demo-
bucket-1a4x26zbcdkqr
        A different default S3 bucket can be set in samconfig.toml and auto resolution
 of buckets turned off by setting resolve_s3=False

        Parameter "stack_name=sam-app" in [default.deploy.parameters] is defined as a
 global parameter [default.global.parameters].
        This parameter will be only saved under [default.global.parameters] in /
Users/.../sam-app/samconfig.toml.

        Saved arguments to config file
        Running 'sam deploy' for future deployments will use the parameters saved
 above.
        The above parameters can be changed by modifying samconfig.toml
        Learn more about samconfig.toml syntax at
        https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/
serverless-sam-cli-config.html

e95fc5e75742: Pushed
d8df51e7bdd7: Pushed
b1d0d7e0b34a: Pushed
0071317b94d8: Pushed
d98f98baf147: Pushed
2d244e0816c6: Pushed
eb2eeb1ebe42: Pushed
a5ca065a3279: Pushed
fe9e144829c9: Pushed
```

```
helloworldfunction-d2f5180b2154-python3.9-v1: digest:
 sha256:cceb71401b47dc3007a7a1e1f2e0baf162999e0e6841d15954745ecc0c447533 size: 2206

        Deploying with following values
        ===============================
        Stack name                   : sam-app
        Region                       : us-west-2
        Confirm changeset            : True
        Disable rollback             : False
        Deployment image repository  :
                                       {
                                           "HelloWorldFunction":
 "012345678910.dkr.ecr.us-west-2.amazonaws.com/samapp7427b055/
helloworldfunction19d43fc4repo"
                                       }
        Deployment s3 bucket         : aws-sam-cli-managed-default-samclisam-s3-demo-
bucket-1a4x26zbcdkqr
        Capabilities                 : ["CAPABILITY_IAM"]
        Parameter overrides          : {}
        Signing Profiles             : {}

Initiating deployment
=====================

HelloWorldFunction may not have authorization defined.
        Uploading to sam-app/682ad27c7cf7a17c7f77a1688b0844f2.template  1328 / 1328
 (100.00%)


Waiting for changeset to be created..

CloudFormation stack changeset
-----------------------------------------------------------------------------------------------
Operation                LogicalResourceId        ResourceType             Replacement

-----------------------------------------------------------------------------------------------
+ Add                    HelloWorldFunctionHell   AWS::Lambda::Permissio   N/A

                         oWorldPermissionProd     n

+ Add                    HelloWorldFunctionRole   AWS::IAM::Role           N/A

+ Add                    HelloWorldFunction       AWS::Lambda::Function    N/A
```

```
+ Add                    ServerlessRestApiDeplo   AWS::ApiGateway::Deplo   N/A

                         yment47fc2d5f9d          yment

+ Add                    ServerlessRestApiProdS   AWS::ApiGateway::Stage   N/A

                         tage

+ Add                    ServerlessRestApi        AWS::ApiGateway::RestA   N/A

                                                  pi

-----------------------------------------------------------------------------------------------


Changeset created successfully. arn:aws:cloudformation:us-
west-2:012345678910:changeSet/samcli-deploy1680634124/0ffd4faf-2e2b-487e-
b9e0-9116e8299ac4


Previewing CloudFormation changeset before deployment
=====================================================
Deploy this changeset? [y/N]: y

2023-04-04 08:49:15 - Waiting for stack create/update to complete

CloudFormation events from stack operations (refresh every 5.0 seconds)
-----------------------------------------------------------------------------------------------
ResourceStatus           ResourceType             LogicalResourceId
 ResourceStatusReason
-----------------------------------------------------------------------------------------------
CREATE_IN_PROGRESS       AWS::CloudFormation::S   sam-app                  User
 Initiated
                         tack

CREATE_IN_PROGRESS       AWS::IAM::Role           HelloWorldFunctionRole   -

CREATE_IN_PROGRESS       AWS::IAM::Role           HelloWorldFunctionRole   Resource
 creation
                                                                           Initiated

CREATE_COMPLETE          AWS::IAM::Role           HelloWorldFunctionRole   -
```

```
CREATE_IN_PROGRESS       AWS::Lambda::Function    HelloWorldFunction       -

CREATE_IN_PROGRESS       AWS::Lambda::Function    HelloWorldFunction       Resource
 creation
                                                                          Initiated

CREATE_COMPLETE          AWS::Lambda::Function    HelloWorldFunction       -

CREATE_IN_PROGRESS       AWS::ApiGateway::RestA   ServerlessRestApi        -

                         pi

CREATE_IN_PROGRESS       AWS::ApiGateway::RestA   ServerlessRestApi        Resource
 creation
                         pi                                                Initiated

CREATE_COMPLETE          AWS::ApiGateway::RestA   ServerlessRestApi        -

                         pi

CREATE_IN_PROGRESS       AWS::Lambda::Permissio   HelloWorldFunctionHell   -

                         n                        oWorldPermissionProd

CREATE_IN_PROGRESS       AWS::ApiGateway::Deplo   ServerlessRestApiDeplo   -

                         yment                    yment47fc2d5f9d

CREATE_IN_PROGRESS       AWS::Lambda::Permissio   HelloWorldFunctionHell   Resource
 creation
                         n                        oWorldPermissionProd     Initiated

CREATE_IN_PROGRESS       AWS::ApiGateway::Deplo   ServerlessRestApiDeplo   Resource
 creation
                         yment                    yment47fc2d5f9d          Initiated

CREATE_COMPLETE          AWS::ApiGateway::Deplo   ServerlessRestApiDeplo   -

                         yment                    yment47fc2d5f9d

CREATE_IN_PROGRESS       AWS::ApiGateway::Stage   ServerlessRestApiProdS   -

                                                  tage
```

```
CREATE_IN_PROGRESS       AWS::ApiGateway::Stage   ServerlessRestApiProdS   Resource
 creation
                                                  tage                     Initiated

CREATE_COMPLETE          AWS::ApiGateway::Stage   ServerlessRestApiProdS   -

                                                  tage

CREATE_COMPLETE          AWS::Lambda::Permissio   HelloWorldFunctionHell   -

                         n                        oWorldPermissionProd

CREATE_COMPLETE          AWS::CloudFormation::S   sam-app                  -

                         tack

-------------------------------------------------------------------------------------------------

CloudFormation outputs from deployed stack
-------------------------------------------------------------------------------------------------
Outputs

-------------------------------------------------------------------------------------------------
Key                 HelloWorldFunctionIamRole

Description         Implicit IAM Role created for Hello World function

Value               arn:aws:iam::012345678910:role/sam-app-HelloWorldFunctionRole-
JFML1JOKHJ71

Key                 HelloWorldApi

Description         API Gateway endpoint URL for Prod stage for Hello World function

Value               https://endlwiqqod.execute-api.us-west-2.amazonaws.com/Prod/hello/


Key                 HelloWorldFunction

Description         Hello World Lambda Function ARN

Value               arn:aws:lambda:us-west-2:012345678910:function:sam-app-
HelloWorldFunction-
```

```
kyg6Y2iNRUPg

------------------------------------------------------------------------------------------------

Successfully created/updated stack - sam-app in us-west-2
```

了解更多

要了解有关使用 Amazon SAM CLI `sam deploy`命令，请参见以下内容：

- 完整 Amazon SAM 研讨会：模块 3-手动部署- 学习如何使用构建、打包和部署无服务器应用程序 Amazon SAM CLI.

用于部署应用程序的选项 Amazon SAM

使用 Amazon SAM，您可以手动部署应用程序，也可以自动部署。使用 Amazon SAM CLI 手动部署您的应用程序。要自动执行部署，请使用管线与持续集成和持续部署（CI/CD）系统。本节中的主题提供有关这两种方法的信息。

主题

- 如何使用 Amazon SAM CLI 手动部署
- 使用 CI/CD 系统和管线进行部署
- 逐步部署
- 使用对部署进行故障排除 Amazon SAM CLI
- 了解更多

如何使用 Amazon SAM CLI 手动部署

在本地开发和测试无服务器应用程序后，您可以使用 sam deploy 命令部署应用程序。

要使用提示 Amazon SAM 指导您完成部署，请指定标--guided志。指定此标记后，sam deploy 命令会压缩应用程序构件，将其上传到 Amazon Simple Storage Service (Amazon S3)（对于 .zip 文件存档）或 Amazon Elastic Container Registry (Amazon ECR)（对于容器映像）。然后，该命令会将您的应用程序部署到 Amazon 云端。

示例：

```
# Deploy an application using prompts:
sam deploy --guided
```

使用 CI/CD 系统和管线进行部署

Amazon SAM 使用管道和持续集成和持续部署 (CI/CD) 系统帮助您实现自动部署。 Amazon SAM 可用于创建管道和简化CI/CD tasks for serverless applications. Multiple CI/CD系统支持 Amazon SAM 构建容器镜像， Amazon SAM 还可以为多个 CI/CD 系统提供一组默认管道模板，这些模板封装了部署最佳实践 Amazon。

有关更多信息，请参阅 使用 CI/CD 系统和管道进行部署 Amazon SAM。

逐步部署

如果您想逐步部署 Amazon SAM 应用程序，而不是一次全部部署，则可以指定 Amazon CodeDeploy 提供的部署配置。有关更多信息，请参阅《Amazon CodeDeploy 用户指南》 CodeDeploy中的使用部署配置。

有关将 Amazon SAM 应用程序配置为逐步部署的信息，请参阅使用以下方法逐步部署无服务器应用程序 Amazon SAM。

使用对部署进行故障排除 Amazon SAM CLI

Amazon SAM CLI 错误：“未满足安全限制”

运行 sam deploy --guided 时，系统用问题 `HelloWorldFunction may not have authorization defined, Is this okay? [y/N]` 向您发出提示。如果您用 **N**（默认选项）来回应此提示，将会出现以下错误：

```
Error: Security Constraints Not Satisfied
```

此提示告知您，您即将部署的应用程序可能在未经授权的情况下配置了 Amazon API Gateway API。用 **N** 来回应此提示，即表明您不同意继续部署。

要解决此问题，您具有以下选项：

- 在经授权的情况下配置应用程序。有关配置授权的更多信息，请参阅 使用您的 Amazon SAM 模板控制 API 访问权限。

- 用 **Y** 来回答此问题，以表明您同意部署在未经授权的情况下配置了 API Gateway API 的应用程序。

了解更多

有关部署无服务器应用程序的实际操作示例，请参阅完整 Amazon SAM 研讨会中的以下内容：

- 模块 3-手动部署- 学习如何使用构建、打包和部署无服务器应用程序 Amazon SAM CLI.
- 模块 4 - CI/CD - 了解如何通过创建持续集成和持续交付 (CI/CD) 管道实现构建、打包和部署阶段的自动化。

使用 CI/CD 系统和管道进行部署 Amazon SAM

Amazon SAM 帮助组织以最少的工作量创建首选CI/CD systems, so that they can realize the benefits of CI/CD管道，例如加快部署频率、缩短变更交付时间和减少部署错误。

Amazon SAM 借助构建容器映像，简化无服务器应用程序的 CI/CD 任务。 Amazon SAM 提供的图像包括 Amazon SAM CLI 并为许多支持的 Amazon Lambda 运行时构建工具。这样可以更轻松地使用以下方法构建和打包无服务器应用程序 Amazon SAM CLI。 这些映像还可以缓解团队为 CI/CD 系统创建和管理自己的映像的需求。有关 Amazon SAM 构建容器镜像的更多信息，请参阅的图像存储库 Amazon SAM。

多个 CI/CD 系统支持 Amazon SAM 构建容器镜像。您应使用哪个 CI/CD 系统取决于多个因素。这些因素包括：应用程序使用单个运行时系统还是多个运行时系统；您是要在容器映像中还是直接在主机（虚拟机 (VM) 或裸机主机）上构建应用程序。

Amazon SAM 还为多个 CI/CD 系统提供了一组默认管道模板，这些模板封装了部署最佳 Amazon实践。这些默认管道模板使用标准的 JSON/YAML 管道配置格式，内置的最佳实践有助于执行多账户和多区域部署，并确保管道不会对基础设施进行意外更改。

您可以使用两个主要选项 Amazon SAM 来部署无服务器应用程序：1) 修改现有的管道配置以使用 Amazon SAM CLI 命令，或 2) 生成示例 CI/CD 管道配置，您可以将其用作自己应用程序的起点。

主题

- 什么是管线？
- 如何在 Amazon SAM 部署时上传本地文件
- 使用以下命令生成入门 CI/CD 管道 Amazon SAM

- 如何使用自定义入门管道 Amazon SAM
- 自动部署 Amazon SAM 应用程序
- 如何对管道使用 OIDC 身份验证 Amazon SAM

什么是管线？

管线是一系列自动执行的步骤，用于发布应用程序的新版本。使用 Amazon SAM，您可以使用许多常见的 CI/CD 系统来部署应用程序，包括 Amazon CodePipelineJenkins、C GitLab I/CD 和 Actions。GitHub

管道模板包括 Amazon 部署最佳实践，可帮助进行多账户和多区域部署。 Amazon 诸如开发和生产环境之类的环境通常存在于不同的 Amazon 账户中。这允许开发团队配置安全的部署管线，而无需对基础设施进行意外更改。

您还可以提供自己的自定义管线模板，以帮助跨开发团队标准化管线。

如何在 Amazon SAM 部署时上传本地文件

将应用程序部署到时 Amazon Web Services 云， Amazon CloudFormation 要求先将本地文件上传到可访问的 Amazon 服务，例如 Amazon Service (Amazon S3)。其中包括您的 Amazon SAM 模板引用的本地文件。为了满足此要求，在使用 `sam deploy` 或 `sam package` 命令时， Amazon SAMCLI 会执行以下操作：

1. 自动将您的本地文件上传到可访问的 Amazon 服务。
2. 自动更新应用程序模板以引用新的文件路径。

主题

- 演示：使用 Amazon SAM CLI 上传 Lambda 函数代码
- 支持的使用案例
- 了解更多

演示：使用 Amazon SAM CLI 上传 Lambda 函数代码

在这个演示中，我们使用 Lambda 函数的 .zip 包类型初始化示例 Hello World 应用程序。我们使用 Amazon SAM CLI 自动将 Lambda 函数代码上传到 Amazon S3，并在应用程序模板中引用其新路径。

首先，我们运行 `sam init` 以初始化 Hello World 应用程序。

```
$ sam init
...
Which template source would you like to use?
        1 - AWS Quick Start Templates
        2 - Custom Template Location
Choice: 1

Choose an AWS Quick Start application template
        1 - Hello World Example
        2 - Multi-step workflow
        ...
Template: 1

Use the most popular runtime and package type? (Python and zip) [y/N]: y

Would you like to enable X-Ray tracing on the function(s) in your application?  [y/
N]: ENTER

Would you like to enable monitoring using CloudWatch Application Insights?
For more info, please view https://docs.aws.amazon.com/AmazonCloudWatch/latest/
monitoring/cloudwatch-application-insights.html [y/N]: ENTER

Project name [sam-app]: demo

    -----------------------
    Generating application:
    -----------------------
    Name: demo
    Runtime: python3.9
    Architectures: x86_64
    Dependency Manager: pip
    Application Template: hello-world
    Output Directory: .
    Configuration file: demo/samconfig.toml

...
```

我们的 Lambda 函数代码组织在项目的 `hello_world` 子目录中。

```
demo
### README.md
```

```
### hello_world
#   ### __init__.py
#   ### app.py
#   ### requirements.txt
### template.yaml
### tests
```

在 Amazon SAM 模板中，我们使用属性引用 Lambda 函数代码的本地路径。`CodeUri`

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
...
Resources:
  HelloWorldFunction:
    Type: AWS::Serverless::Function # More info about Function Resource:
 https://github.com/awslabs/serverless-application-model/blob/master/
versions/2016-10-31.md#awsserverlessfunction
    Properties:
      CodeUri: hello_world/
      Handler: app.lambda_handler
      Runtime: python3.9
      ...
```

接下来，运行 `sam build` 以构建应用程序并准备部署。

```
$ sam build
Starting Build use cache
Manifest file is changed (new hash: 3298f13049d19cffaa37ca931dd4d421) or dependency
 folder (.aws-sam/deps/7896875f-9bcc-4350-8adb-2c1d543627a1) is missing for
 (HelloWorldFunction), downloading dependencies and copying/building source
Building codeuri: /Users/.../demo/hello_world runtime: python3.9 metadata: {}
 architecture: x86_64 functions: HelloWorldFunction
Running PythonPipBuilder:CleanUp
Running PythonPipBuilder:ResolveDependencies
Running PythonPipBuilder:CopySource
Running PythonPipBuilder:CopySource

Build Succeeded

Built Artifacts  : .aws-sam/build
Built Template   : .aws-sam/build/template.yaml
...
```

然后，运行 sam deploy --guided 部署应用程序。

```
$ sam deploy --guided

Configuring SAM deploy
======================

        Looking for config file [samconfig.toml] :  Found
        Reading default arguments  :  Success

        Setting default arguments for 'sam deploy'
        =========================================
        Stack Name [demo]: ENTER
        AWS Region [us-west-2]: ENTER
        #Shows you resources changes to be deployed and require a 'Y' to initiate
 deploy
        Confirm changes before deploy [Y/n]: n
        #SAM needs permission to be able to create roles to connect to the resources in
 your template
        Allow SAM CLI IAM role creation [Y/n]: ENTER
        #Preserves the state of previously provisioned resources when an operation
 fails
        Disable rollback [y/N]: ENTER
        HelloWorldFunction may not have authorization defined, Is this okay? [y/N]: y
        Save arguments to configuration file [Y/n]: ENTER
        SAM configuration file [samconfig.toml]: ENTER
        SAM configuration environment [default]: ENTER

        Looking for resources needed for deployment:
        ...
        Saved arguments to config file
        Running 'sam deploy' for future deployments will use the parameters saved
 above.
        The above parameters can be changed by modifying samconfig.toml
        Learn more about samconfig.toml syntax at
        https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/
serverless-sam-cli-config.html

File with same data already exists at demo/da3c598813f1c2151579b73ad788cac8, skipping
 upload

        Deploying with following values
        ===============================
        Stack name                   : demo
```

```
        Region                      : us-west-2
        Confirm changeset           : False
        Disable rollback            : False
        Deployment s3 bucket        : aws-sam-cli-managed-default-samclisam-s3-demo-
bucket-1a4x26zbcdkqr
        Capabilities                : ["CAPABILITY_IAM"]
        Parameter overrides         : {}
        Signing Profiles            : {}

Initiating deployment
=====================
...
Waiting for changeset to be created..
CloudFormation stack changeset
-----------------------------------------------------------------------------------------------
Operation                LogicalResourceId        ResourceType             Replacement

-----------------------------------------------------------------------------------------------
+ Add                    HelloWorldFunctionHell   AWS::Lambda::Permissio   N/A

                         oWorldPermissionProd     n

+ Add                    HelloWorldFunctionRole   AWS::IAM::Role           N/A

...
-----------------------------------------------------------------------------------------------

Changeset created successfully. arn:aws:cloudformation:us-
west-2:012345678910:changeSet/samcli-deploy1680906292/1164338d-72e7-4593-a372-
f2b3e67f542f

2023-04-07 12:24:58 - Waiting for stack create/update to complete

CloudFormation events from stack operations (refresh every 5.0 seconds)
-----------------------------------------------------------------------------------------------
ResourceStatus           ResourceType             LogicalResourceId
 ResourceStatusReason
-----------------------------------------------------------------------------------------------
CREATE_IN_PROGRESS       AWS::IAM::Role           HelloWorldFunctionRole   -

CREATE_IN_PROGRESS       AWS::IAM::Role           HelloWorldFunctionRole   Resource
 creation
                                                                           Initiated
```

```
...
-----------------------------------------------------------------------------------------------------
CloudFormation outputs from deployed stack
-----------------------------------------------------------------------------------------------------
Outputs

-----------------------------------------------------------------------------------------------------
Key                 HelloWorldFunctionIamRole

Description         Implicit IAM Role created for Hello World function

Value               arn:aws:iam::012345678910:role/demo-HelloWorldFunctionRole-
VQ4CU7UY7S2K

Key                 HelloWorldApi

Description         API Gateway endpoint URL for Prod stage for Hello World function

Value               https://satnon55e9.execute-api.us-west-2.amazonaws.com/Prod/hello/


Key                 HelloWorldFunction

Description         Hello World Lambda Function ARN

Value               arn:aws:lambda:us-west-2:012345678910:function:demo-

HelloWorldFunction-G14inKTmSQvK

-----------------------------------------------------------------------------------------------------
Successfully created/updated stack - demo in us-west-2
```

在部署期间， Amazon SAM CLI 会自动将我们的 Lambda 函数代码上传到 Amazon S3 并更新模板。我们在 Amazon CloudFormation 控制台中修改后的模板反映了 Amazon S3 存储桶路径。

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
...
Resources:
  HelloWorldFunction:
    Type: AWS::Serverless::Function
    Properties:
```

```
      CodeUri: s3://aws-sam-cli-managed-default-samclisam-s3-demo-bucket-1a4x26zbcdkqr/
demo/da3c598813f1c2151579b73ad788cac8
      Handler: app.lambda_handler
      ...
```

支持的使用案例

Amazon SAMCLI可针对多种文件类型、 Amazon CloudFormation 资源类型和 Amazon CloudFormation 宏自动简化此过程。

文件类型

支持应用程序文件和 Docker 映像。

Amazon CloudFormation 资源类型

以下是支持的资源类型及其属性的列表：

资源	属性
AWS::ApiGateway::RestApi	BodyS3Location
AWS::ApiGatewayV2::Api	BodyS3Location
AWS::AppSync:FunctionConfiguration	CodeS3Location RequestMappingTemplateS3Location ResponseMappingTemplateS3Location
AWS::AppSync::GraphQLSchema	DefinitionS3Location
AWS::AppSync::Resolver	CodeS3Location RequestMappingTemplateS3Location ResponseMappingTemplateS3Location

资源	属性
AWS::CloudFormation::ModuleVersion	ModulePackage
AWS::CloudFormation::ResourceVersion	SchemaHandlerPackage
AWS::ECR::Repository	RepositoryName
AWS::ElasticBeanstalk::ApplicationVersion	SourceBundle
AWS::Glue::Job	Command.ScriptLocation
AWS::Lambda::Function	Code Code.ImageUri
AWS::Lambda::LayerVersion	Content
AWS::Serverless::Api	DefinitionUri
AWS::Serverless::Function	CodeUri ImageUri
AWS::Serverless::GraphQLApi	SchemaUri Function.CodeUri Resolver.CodeUri
AWS::Serverless::HttpApi	DefinitionUri
AWS::Serverless::LayerVersion	ContentUri
AWS::Serverless::StateMachine	DefinitionUri

资源	属性
`AWS::StepFunctions::StateMachine`	`DefinitionS3Location`

Amazon CloudFormation 宏

支持使用 `AWS::Include` 转换宏引用的文件。

了解更多

要了解有关`AWS::Include`变换的更多信息，请参阅《Amazon CloudFormation 用户指南》中的 AWS::Include 变换。

要查看在 Amazon SAM 模板中使用`AWS::Include`转换的示例，请参阅 S erver less Land 上的 API Gateway HTTP API 到 SQS 模式。

使用以下命令生成入门 CI/CD 管道 Amazon SAM

准备好自动部署后，您可以使用其中一个入门级管道模板为您选择使用 Amazon SAM的 CI/CD 系统生成部署管道。您的部署管线是您配置并用于自动部署无服务器应用程序的工具。入门管线模板已预配置，可帮助您快速为无服务器应用程序设置部署管线。

借助入门管线模板，您可以使用 sam pipeline init 命令在几分钟内生成管线。

入门级管道模板使用熟悉的JSON/YAML syntax of the CI/CD系统，并纳入了最佳实践，例如跨多个账户和区域管理工件，以及使用部署应用程序所需的最低权限量。目前， Amazon SAM CLI 支持为 Jenkins、CI/CD Amazon CodePipeline、Actions 和 Bitbucket Pipelin es 生成入门 GitLab CI/CD 管道配置GitHub 。

以下是生成入门管道配置所需执行的高级任务：

1. 创建基础设施资源 — 您的管道需要某些 Amazon 资源，例如具有必要权限的 IAM 用户和角色、Amazon S3 存储桶，以及可选的 Amazon ECR 存储库。
2. 将你的 Git 存储库与 CI/CD 系统（你正在使用的CI/CD system needs to know which Git repository will trigger the pipeline to run. Note that this step may not be necessary, depending on which combination of Git repository and CI/CD系统）连接起来。
3. 生成管道配置 – 此步骤将生成包括两个部署阶段的入门管道配置。
4. 将管道配置提交到 Git 存储库 – 此步骤对于确保 CI/CD 系统知道您的管道配置并在提交更改时运行而言是必需的。

生成入门管道配置并将其提交到 Git 存储库后，每当有人向该存储库提交代码更改时，您的管道就会被触发自动运行。

这些步骤的顺序和每个步骤的详细信息因您的 CI/CD 系统而异：

- 如果您正在使用 Amazon CodePipeline，请参阅正在为 Amazon CodePipeline in 生成入门管道 Amazon SAM。
- 如果你使用的是 Jenkins、C GitLab I/CD、Actions 或 Bitbucket Pipelines，请参 GitHub 阅。用于 Amazon SAM 为 Jenkins、C GitLab I/CD、Actions、Bitbucket Pipelines 生成入 GitHub

正在为 Amazon CodePipeline in 生成入门管道 Amazon SAM

要为生成入门工作流配置 Amazon CodePipeline，请按以下顺序执行以下任务：

1. 创建基础设施资源
2. 生成管道配置
3. 将管道配置提交到 Git
4. 将 Git 存储库与 CI/CD 系统连接起来

> **Note**
>
> 以下过程使用了两个 Amazon SAM CLI 命令，`sam pipeline bootstrap`以及`sam pipeline init`。之所以有两个命令，是为了处理这样的用例：管理员（即需要设置基础设施 Amazon 资源的权限的用户，比如 IAM 用户和角色）比开发人员（即只需要设置单个管道的权限，而不需要所需的基础设施 Amazon 资源的用户）拥有更多的权限。

第 1 步：创建基础设施资源

使用的管道 Amazon SAM 需要某些 Amazon 资源，例如具有必要权限的 IAM 用户和角色、Amazon S3 存储桶，以及可选的 Amazon ECR 存储库。对于管道的每个部署阶段，您都必须拥有一组基础设施资源。

您可以运行以下命令来帮助完成此设置：

```
sam pipeline bootstrap
```

> **Note**
>
> 为管道的每个部署阶段运行上一条命令。

第 2 步：生成管道配置

要生成管道配置，请运行以下命令：

```
sam pipeline init
```

第 3 步：将管道配置提交到 Git 存储库

此步骤对于确保 CI/CD 系统知道您的工作流配置并在提交更改时运行而言是必要的。

第 4 步：将 Git 存储库与 CI/CD 系统连接起来

因为 Amazon CodePipeline 您现在可以通过运行以下命令来创建连接：

```
sam deploy -t codepipeline.yaml --stack-name <pipeline-stack-name> --
capabilities=CAPABILITY_IAM --region <region-X>
```

如果您使用的是 Bitbucket， GitHub 则在之前运行sam deploy命令后，请按照开发者工具控制台用户指南中 “更新待处理的连接” 主题中的 “完成连接” 下的步骤完成连接，完成连接。此外，请存储sam deploy命令输出`CodeStarConnectionArn`中的副本，因为如果您想与之以外的其他分支 Amazon CodePipeline 一起使用，则需要该副本`main`。

配置其他分支

默认情况下， Amazon CodePipeline 使用带的`main`分支 Amazon SAM。如果要使用 `main` 以外的分支，则必须再次运行 sam deploy 命令。请注意，根据使用的 Git 仓库，还可能需要提供 `CodeStarConnectionArn`：

```
# For GitHub and Bitbucket
sam deploy -t codepipeline.yaml --stack-name <feature-pipeline-stack-name> --
capabilities=CAPABILITY_IAM --parameter-overrides="FeatureGitBranch=<branch-name>
 CodeStarConnectionArn=<codestar-connection-arn>"

# For Amazon CodeCommit
```

```
sam deploy -t codepipeline.yaml --stack-name <feature-pipeline-stack-name> --
capabilities=CAPABILITY_IAM --parameter-overrides="FeatureGitBranch=<branch-name>"
```

了解更多

有关设置 CI/CD 管道的动手示例，请参阅完整研讨会 Amazon CodePipeline中的 CI/CD。 Amazon SAM

用于 Amazon SAM 为 Jenkins、C GitLab I/CD、Actions、Bitbucket Pipelines 生成入 GitHub

要为 Jenkins、C GitLab I/CD、Actions 或 Bitbucket Pipelines 生成入门工作流配置，请按以下顺序执行以下任务： GitHub

1. 创建基础设施资源
2. 将 Git 存储库与 CI/CD 系统连接起来
3. 创建凭证对象
4. 生成管道配置
5. 将管道配置提交到 Git 存储库

> **Note**
>
> 以下过程使用了两个 Amazon SAM CLI 命令，`sam pipeline bootstrap`以及`sam pipeline init`。之所以有两个命令，是为了处理这样的用例：管理员（即需要设置基础设施 Amazon 资源的权限的用户，比如 IAM 用户和角色）比开发人员（即只需要设置单个管道的权限，而不需要所需的基础设施 Amazon 资源的用户）拥有更多的权限。

第 1 步：创建基础设施资源

使用的管道 Amazon SAM 需要某些 Amazon 资源，例如具有必要权限的 IAM 用户和角色、Amazon S3 存储桶，以及可选的 Amazon ECR 存储库。对于管道的每个部署阶段，您都必须拥有一组基础设施资源。

您可以运行以下命令来帮助完成此设置：

```
sam pipeline bootstrap
```

> **Note**
>
> 为管道的每个部署阶段运行上一条命令。

您必须为管道的每个部署阶段捕获管道用户的 Amazon 证书（密钥 ID 和私有密钥），因为后续步骤需要这些证书。

第 2 步：将 Git 存储库与 CI/CD 系统连接起来

将 Git 存储库连接到CI/CD system is necessary so that the CI/CD系统即可访问用于构建和部署的应用程序源代码。

> **Note**
>
> 如果您使用以下组合之一，则可以跳过此步骤，因为连接会自动完成：
>
> 1. GitHub 使用 GitHub 存储库执行的操作
> 2. GitLab 带有存储库的 CI/CD GitLab
> 3. 带有 Bitbucket 存储库的 Bitbucket 管线

要将 Git 存储库与 CI/CD 系统连接，请执行以下操作之一：

- 如果您使用的是 Jenkins，请参阅 Jenkins 文档中的“添加分支源”。
- 如果您使用的是 C GitLab I/CD 和除之外的 Git 存储库 GitLab，请参阅 “连接外部存储库” GitLab文档。

第 3 步：创建凭证对象

每个CI/CD system has its own way of managing credentials needed for the CI/CD系统都可以访问您的 Git 存储库。

要创建必需的凭证对象，请执行以下操作之一：

- 如果您使用的是 Jenkins，请创建一个存储密钥 ID 和密钥的“凭证”。请按照配置 Jenkins 部分里使用 Amazon SAM构建 Jenkins 管道博客中的说明进行操作。下一步您将需要用到“凭证 ID”。
- 如果您使用的是 C GitLab I/CD，请创建两个 “受保护变量”，密钥 ID 和密钥各一个。按照GitLab 文档中的说明进行操作 — 下一步需要两个 “变量键”。

- 如果您使用的是 Ac GitHub tions，请创建两个 “加密机密”，密钥和密钥各一个。按照GitHub文档中的说明进行操作——下一步需要两个 “机密名称”。
- 如果您使用的是 Bitbucket 管线，请创建两个“安全变量”，密钥 ID 和密钥各一个。请按照变量和密钥中的说明进行操作。下一步您将需要两个“密钥名称”。

第 4 步：生成管道配置

要生成管道配置，请运行以下命令。您需要输入在上一步中创建的凭证对象：

```
sam pipeline init
```

第 5 步：将管道配置提交到 Git 存储库

此步骤对于确保 CI/CD 系统知道您的工作流配置并在提交更改时运行而言是必要的。

了解更多

有关使用 CI/CD 管道设置 CI/CD 管道的动手示例 GitHub Actions，参见 CI/CD GitHub在 “完整工作 Amazon SAM 坊” 中。

如何使用自定义入门管道 Amazon SAM

作为 CI/CD 管理员，您可能想自定义入门管道模板和相关的指导性提示，从而让组织中的开发人员可以使用它们来创建管道配置。

的 Amazon SAM CLI 在创建入门模板时使用 Cookiecutter 模板。有关 cookiecutter 模板的详细信息，请访问 Cookiecutter。

您还可以自定义提示 Amazon SAM CLI 使用`sam pipeline init`命令创建管道配置时向用户显示。要自定义用户提示，请执行以下操作：

1. 创建 **`questions.json`** 文件 – 该 `questions.json` 文件必须位于项目存储库的根目录中。这是与 `cookiecutter.json` 文件相同的目录。要查看 `questions.json` 文件的架构，请参阅 questions.json.schema。要查看示例 `questions.json` 文件，请参阅 questions.json。
2. 使用 cookiecutter 名称映射问题键 – `questions.json` 文件中的每个对象都需要一个与 cookiecutter 模板中的名称相匹配的键。这个按键匹配就是这样 Amazon SAM CLI 将用户的提示响应映射到千篇一律的模板。要查看此键匹配示例，请参阅本主题后面的 示例文件 部分。
3. 创建 **`metadata.json`** 文件 – 在 `metadata.json` 文件中声明管道将包含的阶段数。阶段数指示 `sam pipeline init` 命令提示多少阶段的信息，或者如果是 `--bootstrap` 选项，则为多少

阶段创建基础架构资源。要查看声明具有两个阶段的管道的示例 `metadata.json` 文件，请参阅 metadata.json。

示例项目

以下是示例项目，每个项目都包含一个 Cookiecutter 模板、一个 `questions.json` 文件和一个 `metadata.json` 文件：

- Jenkins 示例：两阶段 Jenkins 管道模板
- CodePipeline 示例：两阶段 CodePipeline 管道模板

示例文件

以下一组文件显示了 `questions.json` 文件中的问题如何与 Cookiecutter 模板文件中的条目相关联。请注意，这些示例是文件片段，并非完整文件。要查看完整文件的示例，请参阅本主题前面的 示例项目 部分。

示例 **`questions.json`**：

```
{
  "questions": [{
    "key": "intro",
    "question": "\nThis template configures a pipeline that deploys a serverless
 application to a testing and a production stage.\n",
    "kind": "info"
  }, {
    "key": "pipeline_user_jenkins_credential_id",
    "question": "What is the Jenkins credential ID (via Jenkins plugin \"aws-
credentials\") for pipeline user access key?",
    "isRequired": true
  }, {
    "key": "sam_template",
    "question": "What is the template file path?",
    "default": "template.yaml"
  }, {
    ...
```

示例 **`cookiecutter.json`**：

```
{
```

```
  "outputDir": "aws-sam-pipeline",
  "pipeline_user_jenkins_credential_id": "",
  "sam_template": "",
    ...
```

示例 **Jenkinsfile**：

```
pipeline {
  agent any
  environment {
    PIPELINE_USER_CREDENTIAL_ID =
 '{{cookiecutter.pipeline_user_jenkins_credential_id}}'
    SAM_TEMPLATE = '{{cookiecutter.sam_template}}'
    ...
```

自动部署 Amazon SAM 应用程序

在中 Amazon SAM，自动部署 Amazon SAM 应用程序的方式因CI/CD system you are using. For this reason, the examples in this section show you how to configure various CI/CD系统而异，以便在 Amazon SAM 构建容器映像中自动构建无服务器应用程序。这些构建容器镜像可以更轻松地使用以下方法构建和打包无服务器应用程序 Amazon SAM CLI.

根据您使用的 CI/CD 系统，现有 CI/CD 管道使用 Amazon SAM 部署无服务器应用程序的过程略有不同。

以下主题提供了配置您的 CI/CD 系统以在构建容器映像中构建无服务器应用程序的 Amazon SAM 示例：

主题

- 使用与 Amazon CodePipeline 进行部署 Amazon SAM
- 使用 Bitbucket 流水线进行部署 Amazon SAM
- 使用 Jenkins 进行部署 Amazon SAM
- 使用 GitLab CI/CD 进行部署 Amazon SAM
- 使用 GitHub 操作进行部署 Amazon SAM

使用与 Amazon CodePipeline 进行部署 Amazon SAM

要将Amazon CodePipeline管道配置为自动生成和部署 Amazon SAM 应用程序，您的 Amazon CloudFormation 模板和`buildspec.yml`文件必须包含执行以下操作的行：

1. 从可用映像中引用具有必要运行时的构建容器映像。以下示例使用 `public.ecr.aws/sam/build-nodejs20.x` 构建容器映像。
2. 配置管道阶段以运行必要的 Amazon SAM 命令行界面 (CLI) 命令。以下示例运行两个 Amazon SAM CLI 命令:sam build和sam deploy（带有必要的选项）。

此示例假设您已使用声明 Amazon SAM 模板文件中的所有函数和层`runtime: nodejs20.x`。

Amazon CloudFormation 模板片段：

```
  CodeBuildProject:
    Type: AWS::CodeBuild::Project
    Properties:
      Environment:
        ComputeType: BUILD_GENERAL1_SMALL
        Image: public.ecr.aws/sam/build-nodejs20.x
        Type: LINUX_CONTAINER
      ...
```

`buildspec.yml` 片段：

```
version: 0.2
phases:
  build:
    commands:
      - sam build
      - sam deploy --no-confirm-changeset --no-fail-on-empty-changeset
```

有关不同运行时的可用 Amazon Elastic Container Registry (Amazon ECR) 构建容器映像的列表，请参阅 的图像存储库 Amazon SAM。

使用 Bitbucket 流水线进行部署 Amazon SAM

要将 Bitbucket Pipelin e 配置为自动构建和部署 Amazon SAM 应用程序，您的`bitbucket-pipelines.yml`文件必须包含执行以下操作的行：

1. 从可用映像中引用具有必要运行时的构建容器映像。以下示例使用 `public.ecr.aws/sam/build-nodejs20.x` 构建容器映像。
2. 配置管道阶段以运行必要的 Amazon SAM 命令行界面 (CLI) 命令。以下示例运行两个 Amazon SAM CLI 命令:sam build和sam deploy（带有必要的选项）。

此示例假设您已使用声明 Amazon SAM 模板文件中的所有函数和层`runtime: nodejs20.x`。

```
image: public.ecr.aws/sam/build-nodejs20.x

pipelines:
  branches:
    main: # branch name
      - step:
          name: Build and Package
          script:
            - sam build
            - sam deploy --no-confirm-changeset --no-fail-on-empty-changeset
```

有关不同运行时的可用 Amazon Elastic Container Registry (Amazon ECR) 构建容器映像的列表，请参阅 的图像存储库 Amazon SAM。

使用 Jenkins 进行部署 Amazon SAM

要将 Jenkins 管道配置为自动生成和部署 Amazon SAM 应用程序，`Jenkinsfile`必须包含执行以下操作的行：

1. 从可用映像中引用具有必要运行时的构建容器映像。以下示例使用 `public.ecr.aws/sam/build-nodejs20.x` 构建容器映像。
2. 配置管道阶段以运行必要的 Amazon SAM 命令行界面 (CLI) 命令。以下示例运行两个 Amazon SAM CLI 命令:sam build和sam deploy（带有必要的选项）。

此示例假设您已使用声明 Amazon SAM 模板文件中的所有函数和层`runtime: nodejs20.x`。

```
pipeline {
    agent { docker { image 'public.ecr.aws/sam/build-nodejs20.x' } }
    stages {
        stage('build') {
            steps {
                sh 'sam build'
                sh 'sam deploy --no-confirm-changeset --no-fail-on-empty-changeset'
            }
        }
    }
}
```

有关不同运行时的可用 Amazon Elastic Container Registry (Amazon ECR) 构建容器映像的列表，请参阅 的图像存储库 Amazon SAM。

使用 GitLab CI/CD 进行部署 Amazon SAM

要将GitLab管道配置为自动生成和部署 Amazon SAM 应用程序，您的`gitlab-ci.yml`文件必须包含执行以下操作的行：

1. 从可用映像中引用具有必要运行时的构建容器映像。以下示例使用 `public.ecr.aws/sam/build-nodejs20.x` 构建容器映像。
2. 配置管道阶段以运行必要的 Amazon SAM 命令行界面 (CLI) 命令。以下示例运行两个 Amazon SAM CLI 命令:sam build和sam deploy（带有必要的选项）。

此示例假设您已使用声明 Amazon SAM 模板文件中的所有函数和层`runtime: nodejs20.x`。

```
image: public.ecr.aws/sam/build-nodejs20.x
deploy:
  script:
    - sam build
    - sam deploy --no-confirm-changeset --no-fail-on-empty-changeset
```

有关不同运行时的可用 Amazon Elastic Container Registry (Amazon ECR) 构建容器映像的列表，请参阅 的图像存储库 Amazon SAM。

使用 GitHub 操作进行部署 Amazon SAM

要将GitHub管道配置为自动生成和部署 Amazon SAM 应用程序，必须先在主机上安装 Amazon SAM 命令行界面 (CLI)。您可以在 GitHub 工作流程中使用GitHub 操作来帮助完成此设置。

以下示例 GitHub 工作流程使用一系列 GitHub操作设置 Ubuntu 主机，然后运行 Amazon SAM CLI 用于生成和部署 Amazon SAM 应用程序的命令：

```
on:
  push:
    branches:
      - main
jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
```

```
      - uses: actions/checkout@v3
      - uses: actions/setup-python@v3
      - uses: aws-actions/setup-sam@v2
      - uses: aws-actions/configure-aws-credentials@v1
        with:
          aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
          aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
          aws-region: us-east-2
      - run: sam build --use-container
      - run: sam deploy --no-confirm-changeset --no-fail-on-empty-changeset
```

有关不同运行时的可用 Amazon Elastic Container Registry (Amazon ECR) 构建容器映像的列表，请参阅 的图像存储库 Amazon SAM。

如何对管道使用 OIDC 身份验证 Amazon SAM

Amazon Serverless Application Model (Amazon SAM) 支持 OpenID Connect (OIDC) 用户身份验证，支持 Bitbucket、 GitHub Actions、 GitLab 持续集成和持续交付（来自这些平台的CI/CD) platforms. With this support, you can use authorized CI/CD用户账户），用于管理您的无服务器应用程序管道。否则，您需要创建和管理多个 Amazon Identity and Access Management (IAM) 用户来控制对 Amazon SAM 管道的访问权限。

使用管道设置 OIDC Amazon SAM

在`sam pipeline bootstrap`配置过程中，请执行以下操作以在您的 Amazon SAM 管道中设置 OIDC。

1. 当系统提示您选择身份提供商时，请选择 OIDC。
2. 接下来，选择支持的 OIDC 提供商。
3. 输入 OIDC 提供商 URL，以 **`https://`** 开头。

> **Note**
>
> Amazon SAM 生成`AWS::IAM::OIDCProvider`资源类型时会引用此 URL。

4. 接下来，按照提示输入访问所选平台所需的 CI/CD 平台信息。这些详细信息因平台而异，可能包括：
   - OIDC 客户端 ID。
   - 代码存储库名称或通用唯一标识符 (UUID)。

- 与存储库关联的组或组织名称。
- GitHub 代码仓库所属的组织。
- GitHub 存储库名称。
- 部署将发生的分支。

5. Amazon SAM 显示输入的 OIDC 配置的摘要。输入某项设置的数字进行编辑，或按下 Enter 以继续。
6. 当系统提示您确认创建支持输入的 OIDC 连接所需的资源时，按下 Y 以继续。

Amazon SAM 使用提供的配置生成一个承担管道执行角色的`AWS::IAM::OIDCProvider` Amazon CloudFormation 资源。要了解有关此 Amazon CloudFormation 资源类型的更多信息，请参阅Amazon CloudFormation 用户指南OIDCProvider中的 AWS:: IAM::。

> **Note**
> 如果身份提供者 (IdP) 资源已存在于您的中 Amazon Web Services 账户，请 Amazon SAM 引用该资源，而不是创建新资源。

示例

以下是使用管道设置 OIDC 的 Amazon SAM 示例。

```
Select a permissions provider:
    1 - IAM (default)
    2 - OpenID Connect (OIDC)
Choice (1, 2): 2
Select an OIDC provider:
    1 - GitHub Actions
    2 - GitLab
    3 - Bitbucket
Choice (1, 2, 3): 1
Enter the URL of the OIDC provider [https://token.actions.githubusercontent.com]:
Enter the OIDC client ID (sometimes called audience) [sts.amazonaws.com]:
Enter the GitHub organization that the code repository belongs to. If there is no
 organization enter your username instead: my-org
Enter GitHub repository name: testing
Enter the name of the branch that deployments will occur from [main]:
```

```
[3] Reference application build resources
Enter the pipeline execution role ARN if you have previously created one, or we will
 create one for you []:
Enter the CloudFormation execution role ARN if you have previously created one, or we
 will create one for you []:
Please enter the artifact bucket ARN for your Lambda function. If you do not have a
 bucket, we will create one for you []:
Does your application contain any IMAGE type Lambda functions? [y/N]:

[4] Summary
Below is the summary of the answers:
    1 - Account: 123456
    2 - Stage configuration name: dev
    3 - Region: us-east-1
    4 - OIDC identity provider URL: https://token.actions.githubusercontent.com
    5 - OIDC client ID: sts.amazonaws.com
    6 - GitHub organization: my-org
    7 - GitHub repository: testing
    8 - Deployment branch: main
    9 - Pipeline execution role: [to be created]
    10 - CloudFormation execution role: [to be created]
    11 - Artifacts bucket: [to be created]
    12 - ECR image repository: [skipped]
Press enter to confirm the values above, or select an item to edit the value:

This will create the following required resources for the 'dev' configuration:
    - IAM OIDC Identity Provider
    - Pipeline execution role
    - CloudFormation execution role
    - Artifact bucket
Should we proceed with the creation? [y/N]:
```

了解更多

有关将 OIDC 与 Amazon SAM 管道配合使用的更多信息，请参阅。sam pipeline bootstrap

使用简介 sam sync 同步到 Amazon Web Services 云

Amazon Serverless Application Model 命令行界面 (Amazon SAM CLI) `sam sync` 命令提供了快速将本地应用程序更改同步到的选项 Amazon Web Services 云。在开发应用程序时使用 `sam sync` 来：

1. 自动检测本地更改并将其同步到 Amazon Web Services 云。

2. 自定义要同步到 Amazon Web Services 云的本地更改。
3. 在云端为应用程序做好测试和验证准备。

借助 `sam sync`，您可以创建快速开发工作流程，缩短将本地更改同步到云端进行测试和验证所需的时间。

> **Note**
>
> 建议在开发环境中使用 `sam sync` 命令。对于生产环境，我们建议使用 `sam deploy` 或配置持续集成和交付 (CI/CD) 管道。要了解更多信息，请参阅使用部署您的应用程序和资源 Amazon SAM。

该`sam sync`命令是其中的一部分 Amazon SAM Accelerate. Amazon SAM Accelerate提供了一些工具，您可以使用这些工具来加快开发和测试中无服务器应用程序的 Amazon Web Services 云体验。

主题

- 自动检测本地更改并将其同步到 Amazon Web Services 云
- 自定义将哪些本地更改同步到 Amazon Web Services 云
- 为云端应用程序做好测试和验证准备
- sam sync 命令的选项
- 故障排除
- 示例
- 了解更多

自动检测本地更改并将其同步到 Amazon Web Services 云

通过 `--watch` 选项运行 `sam sync` 以开始将应用程序同步到 Amazon Web Services 云。其执行以下操作：

1. 构建应用程序 – 此过程与使用 `sam build` 命令类似。
2. 部署您的应用程序 — Amazon SAM CLI Amazon CloudFormation 使用您的默认设置将您的应用程序部署到。使用以下默认值：
   a. Amazon 可在您的`.aws`用户文件夹中找到凭据和常规配置设置。

b. 应用程序部署设置可在应用程序 `samconfig.toml` 文件中找到。

如果找不到默认值， Amazon SAM CLI 将通知您并退出同步过程。

3. 注意当地的变化 — Amazon SAM CLI 保持运行状态并监视应用程序的本地更改。这就是 `--watch` 选项提供的内容。

   默认情况下，此选项可能处于启用状态。有关默认值，请参阅应用程序的 `samconfig.toml` 文件。以下是 文件示例：

```
...
[default.sync]
[default.sync.parameters]
watch = true
...
```

4. 将@@ 本地更改同步到 Amazon Web Services 云 — 当您进行本地更改时， Amazon SAM CLI Amazon Web Services 云 通过最快的可用方法检测这些更改并将其同步到。根据变更的类型，可能会出现以下情况：

   a. 如果您更新的资源支持 Amazon 服务 APIs，则 Amazon SAM CLI 将使用它来部署您的更改。这会导致快速同步以更新 Amazon Web Services 云中的资源。

   b. 如果您更新的资源不支持 Amazon 服务 APIs， Amazon SAM CLI 将执行部 Amazon CloudFormation 署。这会更新您在 Amazon Web Services 云中的整个应用程序。虽然速度不快，但它确实让您不必手动启动部署。

由于该`sam sync`命令会自动更新您的应用程序 Amazon Web Services 云，因此建议仅在开发环境中使用该命令。运行 `sam sync` 时，系统将要求您确认：

```
**The sync command should only be used against a development stack**.

Confirm that you are synchronizing a development stack.

Enter Y to proceed with the command, or enter N to cancel:
 [Y/n]: ENTER
```

自定义将哪些本地更改同步到 Amazon Web Services 云

提供选项以自定义将哪些本地更改同步到 Amazon Web Services 云中。这可以加快在云端查看本地更改以进行测试和验证所需的时间。

例如，提供仅同步代码更改（例如 Amazon Lambda 函数代码）的`--code`选项。在开发过程中，如果您特别关注 Lambda 代码，这会将您的更改快速发送到云端进行测试和验证。以下是示例：

```
$ sam sync --code --watch
```

要仅同步特定 Lambda 函数或层的代码更改，请使用选项 `--resource-id`。以下是示例：

```
$ sam sync --code --resource-id HelloWorldFunction --resource-id HelloWorldLayer
```

为云端应用程序做好测试和验证准备

`sam sync` 命令会自动找到最快的可用方法来更新 Amazon Web Services 云中的应用程序。这可以加快您的开发和云测试工作流程。通过利用 Amazon 服务 APIs，您可以快速开发、同步和测试支持的资源。有关动手示例，请参阅完整 Amazon SAM 研讨会中的模块 6- Amazon SAM 加速。

sam sync 命令的选项

以下是可用于修改 `sam sync` 命令的一些主要选项。有关全部选项的列表，请参阅 sam sync。

执行一次性 Amazon CloudFormation 部署

使用 `--no-watch` 选项关闭自动同步。以下是示例：

```
$ sam sync --no-watch
```

的 Amazon SAM CLI 将执行一次性 Amazon CloudFormation 部署。此命令将 `sam build` 和 `sam deploy` 命令执行的操作组合在一起。

跳过初始 Amazon CloudFormation 部署

您可以自定义每次运行时`sam sync`是否需要 Amazon CloudFormation 部署。

- 规定`--no-skip-deploy-sync`每次运行时`sam sync`都需要 Amazon CloudFormation 部署。这样可以确保您的本地基础设施与之同步 Amazon CloudFormation，从而防止漂移。使用此选项确实会增加开发和测试工作流程的时间。
- 提供`--skip-deploy-sync`使 Amazon CloudFormation 部署成为可选部署。的 Amazon SAM CLI 会将您的本地 Amazon SAM 模板与已部署的 Amazon CloudFormation 模板进行比较，如果未检

测到更改，则会跳过初始 Amazon CloudFormation 部署。在将本地更改同步到时，跳过 Amazon CloudFormation 部署可以节省时间。 Amazon Web Services 云

如果未检测到任何变化， Amazon SAM CLI 在以下情况下仍将执行 Amazon CloudFormation 部署：

- 如果自上次 Amazon CloudFormation 部署以来已经 7 天或更长时间了。
- 如果检测到大量 Lambda 函数代码更改，则使 Amazon CloudFormation 部署成为更新应用程序的最快方法。

以下是示例：

```
$ sam sync --skip-deploy-sync
```

同步嵌套堆栈中的资源

要同步嵌套堆栈中的资源

1. 使用 `--stack-name` 提供根堆栈。
2. 使用以下格式标识嵌套堆栈中的资源：*`nestedStackId/resourceId`*。
3. 使用 `--resource-id` 在嵌套堆栈中提供资源。

   以下是示例：

   ```
   $ sam sync --code --stack-name sam-app --resource-id myNestedStack/
   HelloWorldFunction
   ```

有关创建嵌套应用程序的更多信息，请参阅 使用嵌套应用程序重用代码和资源 Amazon SAM。

指定要更新的特定 Amazon CloudFormation 堆栈

要指定要更新的特定 Amazon CloudFormation 堆栈，请提供`--stack-name`选项。以下是示例：

```
$ sam sync --stack-name dev-sam-app
```

通过在源文件夹中构建项目来加快构建时间

对于受支持的运行时和构建方法，您可以使用 `--build-in-source` 选项直接在源文件夹中生成项目。默认情况下， Amazon SAM CLI 在临时目录中构建，其中包括复制源代码和项目文件。随`--`

build-in-source着， Amazon SAM CLI 直接在源文件夹中构建，无需将文件复制到临时目录，从而加快构建过程。

有关支持的运行时和构建方法的列表，请参阅 --build-in-source。

指定不会启动同步的文件和文件夹

使用 --watch-exclude 选项指定更新后不会启动同步的所有文件或文件夹。有关此选项的更多信息，请参阅 --watch-exclude。

以下是排除与我们的 HelloWorldFunction 函数关联的 package-lock.json 文件示例：

```
$ sam sync --watch --watch-exclude HelloWorldFunction=package-lock.json
```

运行此命令时， Amazon SAM CLI 将启动同步过程。这包括以下这些：

- 运行 sam build 以构建函数并让应用程序准备进行部署。
- 运行 sam deploy 以部署应用程序。
- 注意应用程序的变化。

当我们修改package-lock.json文件时， Amazon SAM CLI 无法启动同步。当另一个文件被更新时， Amazon SAM CLI 将启动同步，其中将包含该package-lock.json文件。

以下是指定子堆栈的 Lambda 函数示例：

```
$ sam sync --watch --watch-exclude ChildStackA/MyFunction=database.sqlite3
```

故障排除

要排除故障 Amazon SAM CLI，请参阅 Amazon SAM CLI 故障排除。

示例

使用 sam sync 更新 Hello World 应用程序

在此示例中，我们首先初始化示例 Hello World 应用程序。要了解有关此应用程序的更多信息，请参阅教程：使用以下命令部署 Hello World 应用程序 Amazon SAM。

运行 sam sync 会开始构建和部署过程。

```
$ sam sync

The SAM CLI will use the AWS Lambda, Amazon API Gateway, and AWS StepFunctions APIs to
 upload your code without
performing a CloudFormation deployment. This will cause drift in your CloudFormation
 stack.
**The sync command should only be used against a development stack**.

Confirm that you are synchronizing a development stack.

Enter Y to proceed with the command, or enter N to cancel:
 [Y/n]:
Queued infra sync. Waiting for in progress code syncs to complete...
Starting infra sync.
Manifest file is changed (new hash: 3298f13049d19cffaa37ca931dd4d421) or dependency
 folder (.aws-sam/deps/0663e6fe-a888-4efb-b908-e2344261e9c7) is missing for
 (HelloWorldFunction), downloading dependencies and copying/building source
Building codeuri: /Users/.../Demo/sync/sam-app/hello_world runtime: python3.9 metadata:
 {} architecture: x86_64 functions: HelloWorldFunction
Running PythonPipBuilder:CleanUp
Running PythonPipBuilder:ResolveDependencies
Running PythonPipBuilder:CopySource

Build Succeeded

Successfully packaged artifacts and wrote output template to file /var/
folders/45/5ct135bx3fn2551_ptl5g6_80000gr/T/tmpx_5t4u3f.
Execute the following command to deploy the packaged template
sam deploy --template-file /var/folders/45/5ct135bx3fn2551_ptl5g6_80000gr/T/tmpx_5t4u3f
 --stack-name <YOUR STACK NAME>


    Deploying with following values
    ===============================
    Stack name                   : sam-app
    Region                       : us-west-2
    Disable rollback             : False
    Deployment s3 bucket         : aws-sam-cli-managed-default-samclisam-s3-demo-
bucket-1a4x26zbcdkqr
    Capabilities                 : ["CAPABILITY_NAMED_IAM", "CAPABILITY_AUTO_EXPAND"]
    Parameter overrides          : {}
    Signing Profiles             : null
```

```
Initiating deployment
=====================


2023-03-17 11:17:19 - Waiting for stack create/update to complete

CloudFormation events from stack operations (refresh every 0.5 seconds)
---------------------------------------------------------------------------------------------------
ResourceStatus                      ResourceType
 LogicalResourceId                   ResourceStatusReason
---------------------------------------------------------------------------------------------------
CREATE_IN_PROGRESS                  AWS::CloudFormation::Stack          sam-app
                     Transformation succeeded
CREATE_IN_PROGRESS                  AWS::CloudFormation::Stack
 AwsSamAutoDependencyLayerNestedSt   -
                                                                        ack
CREATE_IN_PROGRESS                  AWS::IAM::Role
 HelloWorldFunctionRole              -
CREATE_IN_PROGRESS                  AWS::IAM::Role
 HelloWorldFunctionRole              Resource creation Initiated
CREATE_IN_PROGRESS                  AWS::CloudFormation::Stack
 AwsSamAutoDependencyLayerNestedSt   Resource creation Initiated
                                                                        ack
CREATE_COMPLETE                     AWS::IAM::Role
 HelloWorldFunctionRole              -
CREATE_COMPLETE                     AWS::CloudFormation::Stack
 AwsSamAutoDependencyLayerNestedSt   -
                                                                        ack
CREATE_IN_PROGRESS                  AWS::Lambda::Function
 HelloWorldFunction                  -
CREATE_IN_PROGRESS                  AWS::Lambda::Function
 HelloWorldFunction                  Resource creation Initiated
CREATE_COMPLETE                     AWS::Lambda::Function
 HelloWorldFunction                  -
CREATE_IN_PROGRESS                  AWS::ApiGateway::RestApi
 ServerlessRestApi                   -
CREATE_IN_PROGRESS                  AWS::ApiGateway::RestApi
 ServerlessRestApi                   Resource creation Initiated
CREATE_COMPLETE                     AWS::ApiGateway::RestApi
 ServerlessRestApi                   -
CREATE_IN_PROGRESS                  AWS::ApiGateway::Deployment
 ServerlessRestApiDeployment47fc2d   -
                                                                        5f9d
```

```
CREATE_IN_PROGRESS                  AWS::Lambda::Permission
 HelloWorldFunctionHelloWorldPermi   -
                                                                        ssionProd
CREATE_IN_PROGRESS                  AWS::Lambda::Permission
 HelloWorldFunctionHelloWorldPermi   Resource creation Initiated
                                                                        ssionProd
CREATE_IN_PROGRESS                  AWS::ApiGateway::Deployment
 ServerlessRestApiDeployment47fc2d   Resource creation Initiated
                                                                        5f9d
CREATE_COMPLETE                     AWS::ApiGateway::Deployment
 ServerlessRestApiDeployment47fc2d   -
                                                                        5f9d
CREATE_IN_PROGRESS                  AWS::ApiGateway::Stage
 ServerlessRestApiProdStage          -
CREATE_IN_PROGRESS                  AWS::ApiGateway::Stage
 ServerlessRestApiProdStage          Resource creation Initiated
CREATE_COMPLETE                     AWS::ApiGateway::Stage
 ServerlessRestApiProdStage          -
CREATE_COMPLETE                     AWS::Lambda::Permission
 HelloWorldFunctionHelloWorldPermi   -
                                                                        ssionProd
CREATE_COMPLETE                     AWS::CloudFormation::Stack          sam-app
                      -
-----------------------------------------------------------------------------------------------------------

CloudFormation outputs from deployed stack
-----------------------------------------------------------------------------------------------------------
Outputs
-----------------------------------------------------------------------------------------------------------
Key                 HelloWorldFunctionIamRole
Description         Implicit IAM Role created for Hello World function
Value               arn:aws:iam::012345678910:role/sam-app-HelloWorldFunctionRole-
BUFVMO2PJIYF

Key                 HelloWorldApi
Description         API Gateway endpoint URL for Prod stage for Hello World function
Value               https://pcrx5gdaof.execute-api.us-west-2.amazonaws.com/Prod/hello/

Key                 HelloWorldFunction
Description         Hello World Lambda Function ARN
Value               arn:aws:lambda:us-west-2:012345678910:function:sam-app-
HelloWorldFunction-2PlN6TPTQoco
-----------------------------------------------------------------------------------------------------------
Stack creation succeeded. Sync infra completed.
```

```
Infra sync completed.
CodeTrigger not created as CodeUri or DefinitionUri is missing for ServerlessRestApi.
```

部署完成后，我们修改 `HelloWorldFunction` 代码。的 Amazon SAM CLI 检测到此更改并将我们的应用程序同步到。 Amazon Web Services 云由于 Amazon Lambda 支持 Amazon 服务 APIs，因此会执行快速同步。

```
Syncing Lambda Function HelloWorldFunction...
Manifest is not changed for (HelloWorldFunction), running incremental build
Building codeuri: /Users/.../Demo/sync/sam-app/hello_world runtime: python3.9 metadata:
 {} architecture: x86_64 functions: HelloWorldFunction
Running PythonPipBuilder:CopySource
Finished syncing Lambda Function HelloWorldFunction.
```

接下来，我们在应用程序的 Amazon SAM 模板中修改我们的 API 端点。我们将 `/hello` 改为 `/helloworld`。

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
...
Resources:
  HelloWorldFunction:
    ...
    Properties:
      ...
      Events:
        HelloWorld:
          Type: Api
          Properties:
            Path: /helloworld
            Method: get
```

由于 Amazon API Gateway 资源不支持该 Amazon 服务 API，因此 Amazon SAM CLI 自动执行部 Amazon CloudFormation 署。下面是一个示例输出：

```
Queued infra sync. Waiting for in progress code syncs to complete...
Starting infra sync.
Manifest is not changed for (HelloWorldFunction), running incremental build
Building codeuri: /Users/.../Demo/sync/sam-app/hello_world runtime: python3.9 metadata:
 {} architecture: x86_64 functions: HelloWorldFunction
Running PythonPipBuilder:CopySource
```

```
Build Succeeded

Successfully packaged artifacts and wrote output template to file /var/
folders/45/5ct135bx3fn2551_ptl5g6_80000gr/T/tmpuabo0jb9.
Execute the following command to deploy the packaged template
sam deploy --template-file /var/folders/45/5ct135bx3fn2551_ptl5g6_80000gr/T/tmpuabo0jb9
 --stack-name <YOUR STACK NAME>


    Deploying with following values
    ===============================
    Stack name                   : sam-app
    Region                       : us-west-2
    Disable rollback             : False
    Deployment s3 bucket         : aws-sam-cli-managed-default-samclisam-s3-demo-
bucket-1a4x26zbcdkqr
    Capabilities                 : ["CAPABILITY_NAMED_IAM", "CAPABILITY_AUTO_EXPAND"]
    Parameter overrides          : {}
    Signing Profiles             : null

Initiating deployment
=====================


2023-03-17 14:41:18 - Waiting for stack create/update to complete

CloudFormation events from stack operations (refresh every 0.5 seconds)
-----------------------------------------------------------------------------------------------
ResourceStatus                      ResourceType
 LogicalResourceId                   ResourceStatusReason
-----------------------------------------------------------------------------------------------
UPDATE_IN_PROGRESS                  AWS::CloudFormation::Stack          sam-app
                     Transformation succeeded
UPDATE_IN_PROGRESS                  AWS::CloudFormation::Stack
 AwsSamAutoDependencyLayerNestedSt   -
                                                                         ack
UPDATE_COMPLETE                     AWS::CloudFormation::Stack
 AwsSamAutoDependencyLayerNestedSt   -
                                                                         ack
UPDATE_IN_PROGRESS                  AWS::ApiGateway::RestApi
 ServerlessRestApi                   -
UPDATE_COMPLETE                     AWS::ApiGateway::RestApi
 ServerlessRestApi                   -
```

```
CREATE_IN_PROGRESS                  AWS::ApiGateway::Deployment
 ServerlessRestApiDeployment8cf30e   -
                                                                    d3cd
UPDATE_IN_PROGRESS                  AWS::Lambda::Permission
 HelloWorldFunctionHelloWorldPermi   Requested update requires the
                                                                    ssionProd
                      creation of a new physical

                      resource; hence creating one.
UPDATE_IN_PROGRESS                  AWS::Lambda::Permission
 HelloWorldFunctionHelloWorldPermi   Resource creation Initiated
                                                                    ssionProd
CREATE_IN_PROGRESS                  AWS::ApiGateway::Deployment
 ServerlessRestApiDeployment8cf30e   Resource creation Initiated
                                                                    d3cd
CREATE_COMPLETE                     AWS::ApiGateway::Deployment
 ServerlessRestApiDeployment8cf30e   -
                                                                    d3cd
UPDATE_IN_PROGRESS                  AWS::ApiGateway::Stage
 ServerlessRestApiProdStage          -
UPDATE_COMPLETE                     AWS::ApiGateway::Stage
 ServerlessRestApiProdStage          -
UPDATE_COMPLETE                     AWS::Lambda::Permission
 HelloWorldFunctionHelloWorldPermi   -
                                                                    ssionProd
UPDATE_COMPLETE_CLEANUP_IN_PROGRE   AWS::CloudFormation::Stack      sam-app
                      -
SS
DELETE_IN_PROGRESS                  AWS::Lambda::Permission
 HelloWorldFunctionHelloWorldPermi   -
                                                                    ssionProd
DELETE_IN_PROGRESS                  AWS::ApiGateway::Deployment
 ServerlessRestApiDeployment47fc2d   -
                                                                    5f9d
DELETE_COMPLETE                     AWS::ApiGateway::Deployment
 ServerlessRestApiDeployment47fc2d   -
                                                                    5f9d
UPDATE_COMPLETE                     AWS::CloudFormation::Stack
 AwsSamAutoDependencyLayerNestedSt   -
                                                                    ack
DELETE_COMPLETE                     AWS::Lambda::Permission
 HelloWorldFunctionHelloWorldPermi   -
                                                                    ssionProd
```

```
UPDATE_COMPLETE                     AWS::CloudFormation::Stack          sam-app
                      -
-----------------------------------------------------------------------------------------------------

CloudFormation outputs from deployed stack
-----------------------------------------------------------------------------------------------------
Outputs
-----------------------------------------------------------------------------------------------------
Key                 HelloWorldFunctionIamRole
Description         Implicit IAM Role created for Hello World function
Value               arn:aws:iam::012345678910:role/sam-app-HelloWorldFunctionRole-
BUFVMO2PJIYF

Key                 HelloWorldApi
Description         API Gateway endpoint URL for Prod stage for Hello World function
Value               https://pcrx5gdaof.execute-api.us-west-2.amazonaws.com/Prod/hello/

Key                 HelloWorldFunction
Description         Hello World Lambda Function ARN
Value               arn:aws:lambda:us-west-2:012345678910:function:sam-app-
HelloWorldFunction-2PlN6TPTQoco
-----------------------------------------------------------------------------------------------------



Stack update succeeded. Sync infra completed.

Infra sync completed.
```

了解更多

有关全部 `sam sync` 选项的描述，请参阅 sam sync。

使用监控您的无服务器应用程序 Amazon SAM

部署无服务器应用程序后，您可以对其进行监控以提供有关其运行情况的洞察并检测异常，这有助于进行故障排除。本节提供有关监控无服务器应用程序的详细信息。其中包括如何将 Amazon 配置 CloudWatch 为在检测到异常时通知您的信息。它还提供有关使用日志的信息，包括错误突出显示，以及查看、筛选、获取和跟踪日志的提示。

主题

- 使用 “ CloudWatch 应用程序见解” 监控您的 Amazon SAM 无服务器应用程序
- 使用登录 Amazon SAM

使用 “ CloudWatch 应用程序见解” 监控您的 Amazon SAM 无服务器应用程序

Amazon App CloudWatch lication Insights 可帮助您监控应用程序中的 Amazon 资源，以帮助识别潜在问题。它可以分析 Amazon 资源数据以寻找问题迹象，并构建自动仪表板以将其可视化。您可以将 “ CloudWatch 应用程序见解” 配置为与您的 Amazon Serverless Application Model (Amazon SAM) 应用程序一起使用。要了解有关 CloudWatch 应用程序见解的更多信息，请参阅《亚马逊 CloudWatch 用户指南》中的 “亚马逊 CloudWatch 应用程序见解”。

主题

- 使用配置 CloudWatch 应用程序见解 Amazon SAM
- 后续步骤

使用配置 CloudWatch 应用程序见解 Amazon SAM

通过 Amazon SAM 命令行界面为您的 CloudWatch Amazon SAM 应用程序配置应用程序见解 (Amazon SAM CLI) 或通过您的 Amazon SAM 模板。

通过配置 Amazon SAM CLI

使用初始化应用程序时sam init，请通过交互式流程或使用--application-insights选项激活 CloudWatch Application Insights。

要激活《 CloudWatch 应用程序见解》，请通过 Amazon SAM CLI 交互式流程，出现提示**y**时输入。

```
Would you like to enable monitoring using CloudWatch Application Insights?
For more info, please view https://docs.aws.amazon.com/AmazonCloudWatch/latest/
monitoring/cloudwatch-application-insights.html [y/N]:
```

要使用该--application-insights选项激活 CloudWatch Application Insights，请执行以下操作。

```
sam init --application-insights
```

要了解有关使用 sam init 命令的更多信息，请参阅 sam init。

通过 Amazon SAM 模板进行配置

通过在 Amazon SAM 模板中定义`AWS::ResourceGroups::Group`和`AWS::ApplicationInsights::Application`资源来激活 App CloudWatch lication Insights。

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
...
Resources:
  ApplicationResourceGroup:
    Type: AWS::ResourceGroups::Group
    Properties:
      Name:
        Fn::Join:
        - ''
        - - ApplicationInsights-SAM-
        - Ref: AWS::StackName
      ResourceQuery:
        Type: CLOUDFORMATION_STACK_1_0
  ApplicationInsightsMonitoring:
    Type: AWS::ApplicationInsights::Application
    Properties:
      ResourceGroupName:
        Fn::Join:
          - ''
          - - ApplicationInsights-SAM-
          - Ref: AWS::StackName
        AutoConfigurationEnabled: 'true'
    DependsOn: ApplicationResourceGroup
```

- `AWS::ResourceGroups::Group`— 创建一个组来组织您的 Amazon 资源，以便同时管理和自动执行大量资源的任务。在这里，您可以创建一个资源组以与《 CloudWatch 应用见解》配合使用。有关此资源类型的更多信息，请参阅《Amazon CloudFormation 用户指南》中的 `AWS::ResourceGroups::Group`。
- `AWS::ApplicationInsights::Application`— 为资源组配置 “ CloudWatch 应用程序见解”。有关此资源类型的更多信息，请参阅《Amazon CloudFormation 用户指南》中的 `AWS::ApplicationInsights::Application`。

Amazon CloudFormation 在应用程序部署时，这两个资源都会自动传递给。您可以使用 Amazon SAM 模板中的 Amazon CloudFormation 语法进一步配置 “ CloudWatch 应用程序见解”。有关更多信息，请参阅 Amazon CloudWatch 用户指南中的使用 Amazon CloudFormation 模板。

使用该sam init --application-insights命令时，这两个资源都将在您的 Amazon SAM 模板中自动生成。以下是生成的模板的示例。

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: >
  sam-app-test

  Sample SAM Template for sam-app-test

# More info about Globals: https://github.com/awslabs/serverless-application-model/
blob/master/docs/globals.rst
Globals:
  Function:
    Timeout: 3
    MemorySize: 128

Resources:
  HelloWorldFunction:
    Type: AWS::Serverless::Function # More info about Function Resource:
 https://github.com/awslabs/serverless-application-model/blob/master/
versions/2016-10-31.md#awsserverlessfunction
    Properties:
      CodeUri: hello_world/
      Handler: app.lambda_handler
      Runtime: python3.9
      Architectures:
      - x86_64
      Events:
```

```
        HelloWorld:
          Type: Api # More info about API Event Source: https://github.com/awslabs/
serverless-application-model/blob/master/versions/2016-10-31.md#api
          Properties:
            Path: /hello
            Method: get

  ApplicationResourceGroup:
    Type: AWS::ResourceGroups::Group
    Properties:
      Name:
        Fn::Join:
        - ''
        - - ApplicationInsights-SAM-
        - Ref: AWS::StackName
      ResourceQuery:
       Type: CLOUDFORMATION_STACK_1_0
  ApplicationInsightsMonitoring:
    Type: AWS::ApplicationInsights::Application
    Properties:
      ResourceGroupName:
        Fn::Join:
        - ''
        - - ApplicationInsights-SAM-
        - Ref: AWS::StackName
      AutoConfigurationEnabled: 'true'
    DependsOn: ApplicationResourceGroup

Outputs:
  # ServerlessRestApi is an implicit API created out of Events key under
 Serverless::Function
  # Find out more about other implicit resources you can reference within SAM
  # https://github.com/awslabs/serverless-application-model/blob/master/docs/internals/
generated_resources.rst#api
  HelloWorldApi:
    Description: API Gateway endpoint URL for Prod stage for Hello World function
    Value: !Sub "https://${ServerlessRestApi}.execute-api.${AWS::Region}.amazonaws.com/
Prod/hello/"
  HelloWorldFunction:
    Description: Hello World Lambda Function ARN
    Value: !GetAtt HelloWorldFunction.Arn
  HelloWorldFunctionIamRole:
    Description: Implicit IAM Role created for Hello World function
```

```
    Value: !GetAtt HelloWorldFunctionRole.Arn
```

后续步骤

配置 App CloudWatch lication Insights 后，使用sam buildsam deploy来构建应用程序和部署应用程序。所有支持 CloudWatch 应用程序见解的资源都将配置为用于监控。

- 有关支持的资源列表，请参阅 Amazon CloudWatch 用户指南中的支持的日志和指标。
- 要了解如何访问 CloudWatch 应用程序见解，请参阅 Amazon CloudWatch 用户指南中的访问 CloudWatch 应用程序见解。

使用登录 Amazon SAM

为了简化故障排除， Amazon SAM CLI 有一个名为的命令`sam logs`。此命令允许您从命令行获取 Lambda 函数生成的日志。

> **Note**
>
> 该`sam logs`命令适用于所有 Amazon Lambda 函数，而不仅仅是您部署时使用的函数 Amazon SAM。

按堆栈获取日 Amazon CloudFormation 志

当你的函数是 Amazon CloudFormation 堆栈的一部分时，你可以使用该函数的逻辑 ID 来获取日志：

```
sam logs -n HelloWorldFunction --stack-name mystack
```

通过 Lambda 函数名称获取日志

或者，您可以使用函数名称来获取日志：

```
sam logs -n mystack-HelloWorldFunction-1FJ8PD
```

跟踪日志

添加 `--tail` 选项以等待新日志并在日志到达时查看它们。这在部署期间或在解决生产问题时很有用。

```
sam logs -n HelloWorldFunction --stack-name mystack --tail
```

查看特定时间范围的日志

您可以使用 `-s` 和 `-e` 选项查看特定时间范围内的日志：

```
sam logs -n HelloWorldFunction --stack-name mystack -s '10min ago' -e '2min ago'
```

筛选日志

使用 `--filter` 选项在日志事件中快速查找匹配字词、短语或值的日志：

```
sam logs -n HelloWorldFunction --stack-name mystack --filter "error"
```

在输出中， Amazon SAM CLI 为所有出现的 “错误” 一词加下划线，这样您就可以在日志输出中轻松找到过滤器关键字。

错误突出显示

当您的 Lambda 函数崩溃或超时时， Amazon SAM CLI 用红色突出显示超时消息。这会帮助您轻松找到某个日志输出大流内超时的特定执行。

JSON 整齐打印

如果您的日志消息打印 JSON 字符串， Amazon SAM CLI 自动漂亮地打印 JSON，以帮助您直观地解析和理解 JSON。

Amazon SAM 参考

本节包含 Amazon SAM 参考资料。这包括 Amazon SAM CLI 参考资料，例如参考信息 Amazon SAM CLI 命令和其他 Amazon SAM CLI 信息，例如配置、版本控制和故障排除信息。此外，本节还包括有关 Amazon SAM 规范和 Amazon SAM 模板的参考信息，例如有关连接器、映像存储库和部署的参考信息。

Amazon SAM 规格和 Amazon SAM 模板

该 Amazon SAM 规范是 Apache 2.0 许可下的开源规范。该 Amazon SAM 规范的当前版本可在中找到Amazon SAM 项目和 Amazon SAM 模板。 Amazon SAM 规范附带简化的简短语法，用于定义无服务器应用程序的函数 APIs、事件、配置和权限。

您可以通过 Amazon SAM 应用程序项目目录与 Amazon SAM 规范进行交互，应用程序项目目录是运行sam init命令时创建的文件夹和文件。此目录包含 Amazon SAM 模板，这是定义您的 Amazon 资源的重要文件。该 Amazon SAM 模板是模板的 Amazon CloudFormation 扩展。有关 Amazon CloudFormation 模板的完整参考，请参阅《Amazon CloudFormation 用户指南》中的模板参考。

Amazon SAM CLI 命令参考

Amazon Serverless Application Model 命令行界面 (Amazon SAM CLI) 是一个命令行工具，您可以将其与 Amazon SAM 模板和支持的第三方集成一起使用，以构建和运行您的无服务器应用程序。

你可以使用 Amazon SAM CLI 用于开发、测试无服务器应用程序并将其部署到的 Amazon Web Services 云命令。以下是一些示例 Amazon SAM CLI 命令：

- `sam init`— 如果你是第一次 Amazon SAM CLI 用户，你可以运行不带任何参数的`sam init`命令来创建 Hello World 应用程序。该命令使用您选择的语言生成预配置的 Amazon SAM 模板和示例应用程序代码。
- `sam local invoke` 和 `sam local start-api` - 在将应用程序部署到 Amazon Web Services 云之前，使用这些命令在本地测试应用程序代码。
- `sam logs` - 使用此命令获取 Lambda 函数生成的日志。这有助于您在将应用程序部署到 Amazon Web Services 云之后对其进行测试和调试。
- `sam package` - 使用此命令将应用程序代码和依赖项捆绑到部署包中。需要有部署包才能将应用程序上传到 Amazon Web Services 云。
- `sam deploy` - 使用此命令将无服务器应用程序部署到 Amazon Web Services 云。它创建 Amazon 资源并设置 Amazon SAM 模板中定义的权限和其他配置。

有关安装的说明 Amazon SAM CLI，请参阅 安装 Amazon SAM CLI。

Amazon SAM 策略模板

使用 Amazon SAM，您可以从策略模板列表中进行选择，将 Amazon Lambda 函数的权限范围限定为应用程序使用的资源。有关可用策略模板的列表，请参阅策略模板表。有关策略模板和的一般信息 Amazon SAM，请参阅Amazon SAM策略模板。

主题

- Amazon SAM 项目和 Amazon SAM 模板
- Amazon SAM CLI 命令参考
- Amazon SAM CLI 配置文件
- Amazon SAM 连接器参考
- Amazon SAM策略模板
- 的图像存储库 Amazon SAM
- 遥测中的应用 Amazon SAM CLI
- 在 Amazon SAM 模板中设置和管理资源访问权限

Amazon SAM CLI 命令参考

本节包含以下方面的参考信息 Amazon SAM CLI 命令。其中包括有关用法的详细信息、每个命令可用的不同选项的完整列表以及其他信息。在适用的情况下，其他信息包括参数、环境变量和事件等详细信息。有关详细信息，请参阅每个命令。有关安装的说明 Amazon SAM CLI，请参阅 安装 Amazon SAM CLI。

主题

- sam build
- sam delete
- sam deploy
- sam init
- sam list
- sam local generate-event

- sam local invoke
- sam local start-api
- sam local start-lambda
- sam logs
- sam package
- sam pipeline bootstrap
- sam pipeline init
- sam publish
- sam remote invoke
- sam remote test-event
- sam sync
- sam traces
- sam validate

sam build

本页提供 Amazon Serverless Application Model 命令行界面 (Amazon SAMCLI) `sam build` 命令的参考信息。

- 有关简介 Amazon SAMCLI，请参阅 那是什么 Amazon SAMCLI？
- 有关使用该 Amazon SAMCLI`sam build`命令的文档，请参阅建筑简介 Amazon SAM。

`sam build` 命令用于让应用程序为开发工作流程的后续步骤（例如，本地测试或部署到 Amazon Web Services 云）做好准备。

使用量

```
$ sam build <arguments> <options>
```

参数

资源 ID

可选。指示生成 Amazon SAM 在Amazon SAM 模板中声明的单个资源。指定资源的构建构件将是唯一可用于工作流程中后续命令的构建构件，即 `sam package` 和 `sam deploy`。

选项

`--base-dir, -s DIRECTORY`

根据此目录解析函数或层的源代码的相对路径。如果要更改源代码文件夹相对路径的解析方式，请使用此选项。默认情况下，相对路径是根据 Amazon SAM 模板的位置进行解析的。

除了正在构建的根应用程序或堆栈中的资源外，此选项还应用嵌套应用程序或堆栈。

此选项适用于以下资源类型和属性：

- 资源类型：`AWS::Serverless::Function` 属性：`CodeUri`
- 资源类型：`AWS::Serverless::Function` 资源属性：`Metadata` 条目：`DockerContext`
- 资源类型：`AWS::Serverless::LayerVersion` 属性：`ContentUri`
- 资源类型：`AWS::Lambda::Function` 属性：`Code`
- 资源类型：`AWS::Lambda::LayerVersion` 属性：`Content`

`--beta-features | --no-beta-features`

允许或拒绝测试版功能。

`--build-dir, -b DIRECTORY`

已构建的构件的存储目录路径。使用此选项可移除此目录及其所有内容。

`--build-image TEXT`

您要为构建提取容器映像的 URI。默认情况下， Amazon SAM 从 Amazon ECR Public 提取容器映像。使用此选项可从其他位置提取映像。

您可以多次指定该选项。此选项的每个实例都可以采用字符串或键值对。如果指定字符串，则字符串就是要用于应用程序中所有资源的容器映像 URI。例如 `sam build --use-container --build-image amazon/aws-sam-cli-build-image-python3.8`。如果指定键值对，则键是资源名称，值是要用于该资源的容器映像 URI。例如 `sam build --use-container --build-image Function1=amazon/aws-sam-cli-build-image-python3.8`。如果使用键值对，您可以为不同的资源指定不同的容器映像。

在指定了 `--use-container` 选项的情况下此选项才适用，否则会导致错误。

`--build-in-source | --no-build-in-source`

提供 `--build-in-source` 以便直接在源文件夹中生成项目。

`--build-in-source` 选项支持以下运行时和构建方法：

- 运行时：`sam init --runtime` 选项支持的任何 Node.js 运行时。
- 构建方法：`Makefile`、`esbuild`。

`--build-in-source` 选项与以下选项不兼容：

- `--hook-name`
- `--use-container`

默认值：`--no-build-in-source`

`--cached | --no-cached`

启用或禁用缓存的构建。使用此选项可以重复使用与之前版本相比未更改的构建工件。 Amazon SAM 评估您是否更改了项目目录中的任何文件。默认情况下，不会缓存构建。如果调用了该`--no-cached`选项，它将覆盖 samconfig.t `cached = true` oml 中的设置。

> **Note**
>
> Amazon SAM 不会评估在您未提供特定版本的情况下，您是否更改了项目所依赖的第三方模块。例如，如果您的 Python 函数包含带有该条目的`requirements.txt`文件`requests=1.x`，并且最新的请求模块版本从更改`1.1`为`1.2`，则在运行非缓存版本之前， Amazon SAM 不会提取最新版本。

`--cache-dir`

在指定了 `--cached` 的情况下用于存储缓存构件的目录。默认缓存目录为 `.aws-sam/cache`。

`--config-env` *TEXT*

在配置文件中指定要使用的默认参数值的环境名称。默认值为“default”。有关配置文件的详细信息，请参阅 Amazon SAM CLI 配置文件。

`--config-file` *PATH*

包含要使用的默认参数值的配置文件的路径和文件名。在项目目录的根目录中，默认值为“`samconfig.toml`”。有关配置文件的详细信息，请参阅 Amazon SAM CLI 配置文件。

`--container-env-var, -e` *TEXT*

要传递到构建容器的环境变量。您可以多次指定该选项。此选项的每个实例都采用键值对，其中键是资源和环境变量，值是环境变量的值。例如：`--container-env-var Function1.GITHUB_TOKEN=TOKEN1 --container-env-var Function2.GITHUB_TOKEN=TOKEN2`。

在指定了 `--use-container` 选项的情况下此选项才适用，否则会导致错误。

`--container-env-var-file, -ef` *PATH*

包含容器环境变量值的 JSON 文件的路径和文件名。有关容器环境变量文件的更多信息，请参阅容器环境变量文件。

在指定了 `--use-container` 选项的情况下此选项才适用，否则会导致错误。

`--debug`

启用调试日志记录，以打印 Amazon SAM CLI 生成的调试消息并显示时间戳。

`--docker-network` *TEXT*

指定 Lambda Docker 容器应连接到的现有 Docker 网络的名称或 ID，以及默认桥接网络。如果未指定此项，Lambda 容器将仅连接到默认的桥接 Docker 网络。

`--exclude, -x`

要从 `sam build` 中排除的资源的名称。例如，如果模板包含 `Function1`、`Function2` 和 `Function3`，并且您运行 `sam build --exclude Function2`，则只会构建 `Function1` 和 `Function3`。

`--help`

显示此消息并退出。

`--hook-name` *TEXT*

用于扩展 Amazon SAM CLI 功能的钩子的名称。

可接受的值：`terraform`。

`--manifest , -m` *PATH*

要使用的自定义依赖项清单文件（例如 package.json）的路径，而不是默认路径。

`--mount-symlinks`

确保 Amazon SAMCLI始终挂载文件中存在的符号链接以进行构建或调用。这仅适用于顶级目录中的符号链接（即直接位于函数根目录的符号链接）。默认情况下，除了需要在 NodeJS `node_modules` 中使用的`build-in-source`符号链接外，不会挂载符号链接。

`--no-use-container`

允许您使用 IDE 工具包设置默认行为的选项。您也可以使用`sam build --no-use-container`在本地计算机上运行构建，而不是 docker 容器。

`--parallel`

启用并行构建。使用此选项并行构建 Amazon SAM 模板的函数和图层。默认情况下，函数和层是按顺序构建的。

`--parameter-overrides`

（可选）包含编码为键值对的 Amazon CloudFormation 参数覆盖的字符串。使用与 Amazon Command Line Interface (Amazon CLI) 相同的格式。例如：'`ParameterKey=KeyPairName, ParameterValue=MyKey ParameterKey=InstanceType, ParameterValue=t1.micro`'。此选项与 `--hook-name` 不兼容。

`--profile` *`TEXT`*

您的凭证文件中用于获取 Amazon 凭证的特定个人资料。

`--region` *`TEXT`*

Amazon Web Services 区域 要部署到的。例如，us-east-1。

`--save-params`

将您在命令行中提供的参数保存到 Amazon SAM 配置文件中。

`--skip-prepare-infra`

如果没有进行任何基础架构更改，则跳过准备阶段。使用 `--hook-name` 选项。

`--skip-pull-image`

指定命令是否应跳过下拉最新 Docker 映像获取 Lambda 运行时的操作。

`--template-file, --template, -t` *`PATH`*

Amazon SAM 模板文件的路径和文件名`[default: template.[yaml|yml]]`。此选项与 `--hook-name` 不兼容。

`--terraform-project-root-path`

包含 Terraform 配置文件或函数源代码的顶级目录的相对路径或绝对路径。如果这些文件位于包含 Terraform 根模块的目录之外，请使用此选项指定其绝对路径或相对路径。此选项要求将 `--hook-name` 设置为 `terraform`。

`--use-container, -u`

如果函数依赖于具有本地编译的依赖项的程序包，请使用此选项在类似于 Lambda 的 Docker 容器中构建函数。

示例

有关使用 `sam build` 子命令的详细示例和深入演练，请参阅建筑简介 Amazon SAM。

sam delete

本页提供 Amazon Serverless Application Model 命令行界面的参考信息 (Amazon SAM CLI) `sam delete` 命令。

有关简介 Amazon SAM CLI，请参阅 那是什么 Amazon SAMCLI？

该`sam delete`命令通过删除 Amazon CloudFormation 堆栈、打包并部署到 Amazon S3 和 Amazon ECR 的项目以及 Amazon SAM 模板文件来删除 Amazon SAM 应用程序。

此命令还会检查是否部署了 Amazon ECR 配套堆栈，如果是，则会提示用户删除该堆栈和 Amazon ECR 存储库。如果指定了 `--no-prompts`，则默认情况下会删除配套堆栈和 Amazon ECR 存储库。

使用量

```
$ sam delete <options>
```

选项

`--config-env` *TEXT*

在配置文件中指定要使用的默认参数值的环境名称。默认值为 `default`。有关配置文件的详细信息，请参阅 Amazon SAM CLI 配置文件。

`--config-file` *PATH*

包含要使用的默认参数值的配置文件的路径和文件名。在项目目录的根目录中，默认值为 `samconfig.toml`。有关配置文件的详细信息，请参阅 Amazon SAM CLI 配置文件。

`--debug`

开启调试日志记录以打印调试消息 Amazon SAM CLI 生成并显示时间戳。

`--help`

显示此消息并退出。

`--no-prompts`

将此选项指定为在非交互模式下 Amazon SAM 操作。必须使用 `--stack-name` 选项或在 `toml` 配置文件中提供堆栈名称。

`--profile` *TEXT*

您的凭证文件中用于获取 Amazon 凭证的特定个人资料。

`--region` *TEXT*

要部署到的 Amazon 区域。例如，us-east-1。

`--s3-bucket`

要删除的 Amazon S3 存储桶的路径。

`--s3-prefix`

要删除的 Amazon S3 存储桶的前缀。

`--save-params`

将您在命令行中提供的参数保存到 Amazon SAM 配置文件中。

`--stack-name` *TEXT*

要删除的 Amazon CloudFormation 堆栈的名称。

示例

以下命令删除堆栈 `MY-STACK`。

```
$ sam delete --stack-name MY-STACK
```

以下命令删除堆栈 `MY-STACK` 和 S3 存储桶 `sam-s3-demo-bucket`：

```
$ sam delete \
    --stack-name MyStack \
    --s3-bucket MySAMBucket
```

sam deploy

本页提供 Amazon Serverless Application Model 命令行界面的参考信息 (Amazon SAM CLI) `sam deploy` 命令。

- 有关简介 Amazon SAM CLI，请参阅 那是什么 Amazon SAMCLI？
- 有关使用方法的文档 Amazon SAM CLI `sam deploy`命令，请参阅使用部署简介 Amazon SAM。

该sam deploy命令向用户部署应用程序。 Amazon Web Services 云 Amazon CloudFormation

使用量

```
$ <environment variables> sam deploy <options>
```

环境变量

SAM_CLI_POLL_DELAY

在 shell 中将SAM_CLI_POLL_DELAY环境变量设置为秒值，以配置 Amazon SAM CLI 检查 Amazon CloudFormation 堆栈状态的频率，这在查看来自的限制时很有用。 Amazon CloudFormation此 env 变量用于轮询 describe_stack API 调用，这些调用是在运行 sam deploy 时进行的。

以下是此变量的示例：

```
$ SAM_CLI_POLL_DELAY=5 sam deploy
```

选项

--capabilities *LIST*

必须指定才能创建特定堆栈 Amazon CloudFormation 的功能列表。某些堆栈模板可能包含影响您权限的资源 Amazon Web Services 账户，例如，通过创建新 Amazon Identity and Access Management (IAM) 用户来影响您的权限。对于这些堆栈，您必须通过指定此选项来明确确认它们的功能。有效值仅为 CAPABILITY_IAM 和 CAPABILITY_NAMED_IAM。如果有 IAM 资源，则您可以指定任意一个功能。如果有具有自定义名称的 IAM 资源，则必须指定 CAPABILITY_NAMED_IAM。如果不指定此选项，则操作会返回 InsufficientCapabilities 错误。

部署包含嵌套应用程序的应用程序时，必须使用CAPABILITY_AUTO_EXPAND来确认该应用程序包含嵌套应用程序。有关更多信息，请参阅 部署嵌套应用程序。

--config-env *TEXT*

在配置文件中指定要使用的默认参数值的环境名称。默认值为 default。有关配置文件的详细信息，请参阅 Amazon SAM CLI 配置文件。

`--config-file` *`PATH`*

包含要使用的默认参数值的配置文件的路径和文件名。在项目目录的根目录中，默认值为 `samconfig.toml`。有关配置文件的详细信息，请参阅 Amazon SAM CLI 配置文件。

`--confirm-changeset | --no-confirm-changeset`

提示确认是否 Amazon SAM CLI 部署计算出的变更集。

`--debug`

打开调试日志以打印调试消息 Amazon SAM CLI 生成并显示时间戳。

`--disable-rollback | --no-disable-rollback`

指定在部署期间发生错误时是否回滚 Amazon CloudFormation 堆栈。默认情况下，如果部署期间出现错误，您的 Amazon CloudFormation 堆栈会回滚到上一个稳定状态。如果指定 `--disable-rollback`，当部署期间出现错误时，则在错误出现之前创建或更新的资源不会回滚。

`--fail-on-empty-changeset | --no-fail-on-empty-changeset`

指定在未对堆栈进行任何更改时是否要返回非零退出代码。默认行为是返回非零退出代码。

`--force-upload`

指定此选项可上传构件，即使要上传的构件与 Amazon S3 存储桶中的现有构件匹配。匹配的构件会被覆盖。

`--guided, -g`

指定此选项可获得 Amazon SAM CLI 使用提示指导您完成部署。

`--help`

显示此消息并退出。

`--image-repositories` *`TEXT`*

函数与其 Amazon ECR 存储库 URI 之间的映射。通过逻辑 ID 引用函数。以下是示例：

```
$ sam deploy --image-repositories Function1=123456789012.dkr.ecr.us-
east-1.amazonaws.com/my-repo
```

您可以在单个命令中多次指定此选项。

`--image-repository` *TEXT*

此命令用于上传函数映像的 Amazon ECR 存储库的名称。对于使用 `Image` 包类型声明的函数，必须使用此选项。

`--kms-key-id` *TEXT*

Amazon Key Management Service (Amazon KMS) 密钥的 ID，用于对 Amazon S3 存储桶中的静态项目进行加密。如果您未指定此选项，则 Amazon SAM 使用 Amazon S3 托管的加密密钥。

`--metadata`

要附加到模板中引用的所有构件的元数据的映射。

`--no-execute-changeset`

指示是否要应用变更集。如果要在应用变更集之前查看堆栈更改，请指定此选项。此命令会创建 Amazon CloudFormation 变更集，然后退出而不应用变更集。要应用变更集，请在不使用此选项的情况下运行同一命令。

`--no-progressbar`

将构件上传到 Amazon S3 时不显示进度条。

`--notification-arns` *LIST*

与堆栈关联的亚马逊简单通知服务 (Amazon SNS) Service ARNs 主题 Amazon CloudFormation 列表。

`--on-failure [ROLLBACK | DELETE | DO_NOTHING]`

指定堆栈创建失败时要采取的操作。

以下选项可用：

- `ROLLBACK` - 将堆栈回滚到上一个已知良好状态。
- `DELETE` - 将堆栈回滚到上一个已知良好状态（如果存在）。否则，删除堆栈。
- `DO_NOTHING` - 既不回滚堆栈，也不删除堆栈。其效果与 `--disable-rollback` 相同。

默认行为是 `ROLLBACK`。

> **Note**
>
> 您可以指定 `--disable-rollback` 选项或 `--on-failure` 选项，但不能同时指定两者。

`--parameter-overrides` *LIST*

包含编码为键值对的 Amazon CloudFormation 参数覆盖的字符串。使用与 Amazon Command Line Interface (Amazon CLI) 相同的格式。的 Amazon SAM CLI format 是显式的键和值关键字，每个替换都用空格分隔。以下是两个示例：

```
$ sam deploy --parameter-overrides ParameterKey=value1,ParameterValue=value2
```

```
$ sam deploy --parameter-overrides ParameterKey=value1,ParameterValue=value2
 ParameterKey=hello,ParameterValue=world ParameterKey=apple,ParameterValue=banana
```

`--profile` *TEXT*

您的凭证文件中用于获取 Amazon 凭证的特定个人资料。

`--region` *TEXT*

Amazon Web Services 区域 要部署到的。例如，us-east-1。

`--resolve-image-repos`

自动创建 Amazon ECR 存储库，用于打包和部署非指导式部署。此选项仅适用于指定了 `PackageType: Image` 的函数和层。如果您指定了`--guided`选项，那么 Amazon SAM CLI 忽略。`--resolve-image-repos`

> **Note**
>
> 如果使用此选项 Amazon SAM 自动为函数或层创建任何 Amazon ECR 存储库，而您随后从 Amazon SAM 模板中删除了这些函数或层，则相应的 Amazon ECR 存储库将被自动删除。

`--resolve-s3`

自动创建 Amazon S3 存储桶，用于打包和部署非指导式部署。如果指定 `--guided` 选项，则 Amazon SAM CLI 会忽略 `--resolve-s3`。如果同时指定 `--s3-bucket` 和 `--resolve-s3` 选项，则会出现错误。

`--role-arn` *TEXT*

应用变更集时 Amazon CloudFormation 扮演的 IAM 角色的 Amazon 资源名称 (ARN)。

`--s3-bucket` *TEXT*

此命令用于上传您的 Amazon CloudFormation 模板的 Amazon S3 存储桶的名称。如果模板大于 51,200 字节，则需要 `--s3-bucket` 或 `--resolve-s3` 选项。如果同时指定 `--s3-bucket` 和 `--resolve-s3` 选项，则会出现错误。

`--s3-prefix` *TEXT*

上传到 Amazon S3 存储桶的构件的名称中添加的前缀。前缀名称是 Amazon S3 存储桶的路径名称（文件夹名称）。

`--save-params`

将您在命令行中提供的参数保存到 Amazon SAM 配置文件中。

`--signing-profiles` *LIST*

用于签署部署包的签名配置文件的列表。此选项采用键值对列表，其中密钥是要签名的函数或层的名称，值是签名配置文件，可选的配置文件所有者用：分隔。例如，`FunctionNameToSign=SigningProfileName1 LayerNameToSign=SigningProfileName2:SigningProfileOwner`。

`--stack-name` *TEXT*

（必填）您要部署到的 Amazon CloudFormation 堆栈的名称。如果指定现有堆栈，则该命令将更新堆栈。如果指定新堆栈，则该命令将创建它。

`--tags` *LIST*

要与已创建或更新的堆栈关联的标签列表。 Amazon CloudFormation 还会将这些标签传播到堆栈中支持它的资源。

`--template-file, --template, -t` *PATH*

您的 Amazon SAM 模板所在的路径和文件名。

> **Note**
>
> 如果指定此选项，则仅 Amazon SAM 部署模板及其指向的本地资源。

`--use-json`

为 Amazon CloudFormation 模板输出 JSON。默认输出为 YAML。

示例

有关使用 `sam deploy` 子命令的详细示例和深入演练，请参阅使用部署简介 Amazon SAM。

sam init

本页提供 Amazon Serverless Application Model 命令行界面的参考信息 (Amazon SAM CLI) `sam init` 命令。

- 有关简介 Amazon SAM CLI，请参阅 那是什么 Amazon SAMCLI？
- 有关使用方法的文档 Amazon SAM CLI `sam init`命令，请参阅在中创建您的应用程序 Amazon SAM。

`sam init` 命令提供用于初始化新的无服务器应用程序的选项。

使用量

```
$ sam init <options>
```

选项

`--app-template` *`TEXT`*

您要使用的托管应用程序模板的标识符。如果您不确定，请在没有交互式工作流选项的情况下调用 `sam init`。

如果指定了 `--no-interactive`，且不提供 `--location` ，则需要此参数。

此参数仅在以下版本中可用 Amazon SAM CLI 版本 0.30.0 及更高版本。对早期版本指定此参数会导致错误。

`--application-insights | --no-application-insights`

为您的 CloudWatch 应用程序激活 Amazon 应用程序见解监控。要了解更多信息，请参阅使用 “CloudWatch 应用程序见解” 监控您的 Amazon SAM 无服务器应用程序。

默认选项是 `--no-application-insights`。

`--architecture, -a` *`[ x86_64 | arm64 ]`*

应用程序的 Lambda 函数的指令集架构。指定 `x86_64` 或 `arm64`。

`--base-image` *[amazon/dotnet8-base | amazon/dotnet6-base | amazon/java21-base | amazon/java17-base | amazon/java11-base | amazon/nodejs22.x-base | amazon/nodejs20.x-base | amazon/nodejs18.x-base | amazon/nodejs16.x-base | amazon/python3.13-base | amazon/python3.12-base | amazon/python3.11-base | amazon/python3.10-base | amazon/python3.9-base | amazon/python3.8-base | amazon/ruby3.4-base | amazon/ruby3.3-base | amazon/ruby3.2-base]*

应用程序的基本映像。当软件包类型为 `Image` 时，此选项才适用。

如果已指定 `--no-interactive`，`--package-type` 被指定为 `Image`，且未指定 `--location`，则需要此参数。

`--config-env` *TEXT*

在配置文件中指定要使用的默认参数值的环境名称。默认值为“default”。有关配置文件的详细信息，请参阅 Amazon SAM CLI 配置文件。

`--config-file` *PATH*

包含要使用的默认参数值的配置文件的路径和文件名。在项目目录的根目录中，默认值为“samconfig.toml”。有关配置文件的详细信息，请参阅 Amazon SAM CLI 配置文件。

`--debug`

开启调试日志记录以打印调试消息 Amazon SAM CLI 生成，并显示时间戳。

`--dependency-manager, -d` *[gradle | mod | maven | bundler | npm | cli-package | pip]*

Lambda 运行时的依赖项管理器。

`--extra-content`

覆盖模板的 `cookiecutter.json` 配置中的任何自定义参数，例如 `{"customParam1": "customValue1", "customParam2":"customValue2"}`。

`--help, -h`

显示此消息并退出。

`--location, -l` *TEXT*

模板或应用程序的位置（Git、Mercurial、HTTP/HTTPS、.zip 文件、路径）。

如果指定了 `--no-interactive` 且未提供 `--runtime`、`--name` 和 `--app-template`，则需要此参数。

对于 Git 存储库，必须使用存储库根目录的位置。

对于本地路径，模板必须为 .zip 文件或 Cookiecutter 格式。

`--name, -n TEXT`

要生成为目录的项目的名称。

如果指定了 `--no-interactive`，且不提供 `--location` ，则需要此参数。

`--no-input`

禁用 Cookiecutter 提示并接受模板配置中定义的 vcfdefault 值。

`--no-interactive`

禁用 init 参数的交互式提示，如果缺少任何必需值，则失败。

`--output-dir, -o PATH`

已初始化的应用程序的输出位置。

`--package-type [ Zip | Image ]`

示例应用程序的软件包类型。`Zip` 会创建 .zip 文件存档，`Image` 会创建容器映像。

`--runtime, -r [ dotnet8 | dotnet6 | java21 | java17 | java11 | nodejs22.x | nodejs20.x | nodejs18.x | nodejs16.x | python3.13 | python3.12 | python3.11 | python3.10 | python3.9 | python3.8 | ruby3.4 | ruby3.3 | ruby3.2 ]`

应用程序的 Lambda 运行时。当软件包类型为 `Zip` 时，此选项才适用。

如果已指定 `--no-interactive`，`--package-type` 被指定为 `Zip`，且未指定 `--location`，则需要此参数。

`--save-params`

将您在命令行中提供的参数保存到 Amazon SAM 配置文件中。

`--tracing | --no-tracing`

激活您的 Lambda 函数的 Amazon X-Ray 跟踪。

示例

有关使用 `sam init` 子命令的详细示例和深入演练，请参阅在中创建您的应用程序 Amazon SAM。

sam list

本页提供 Amazon Serverless Application Model 命令行界面的参考信息 (Amazon SAM CLI) `sam list` 命令。

有关简介 Amazon SAM CLI，请参阅 那是什么 Amazon SAMCLI？

`sam list` 命令输出有关无服务器应用程序中资源和无服务器应用程序状态的重要信息。在部署之前和之后使用 sam list，以在本地开发和云开发期间提供帮助。

使用量

```
$ sam list <options> <subcommand>
```

选项

`--help, -h`

显示此消息并退出。

子命令

`endpoints`

显示堆 Amazon CloudFormation 栈中的云端和本地端点列表。有关更多信息，请参阅 sam list endpoints。

`resources`

显示部署时在您的 Amazon Serverless Application Model (Amazon SAM) 模板中创建 Amazon CloudFormation 的资源。有关更多信息，请参阅 sam list resources。

`stack-outputs`

显示来自 Amazon SAM 或 Amazon CloudFormation 模板的 Amazon CloudFormation 堆栈输出。有关更多信息，请参阅 sam list stack-outputs。

sam list endpoints

本页提供 Amazon Serverless Application Model 命令行界面的参考信息 (Amazon SAM CLI) `sam list endpoints` 子命令。

有关简介 Amazon SAM CLI，请参阅 那是什么 Amazon SAMCLI？

`sam list endpoints`子命令显示 Amazon CloudFormation 堆栈中的云端和本地端点列表。您可以通过 sam local 和 sam sync 命令与这些资源进行交互。

Amazon Lambda 此命令支持 Amazon API Gateway 资源类型。

> **Note**
>
> 如果为 Amazon API Gateway 资源配置了自定义域，则这些域受支持。此命令会输出自定义域而不是默认端点。

使用量

```
$ sam list endpoints <options>
```

选项

`--config-env` *TEXT*

在配置文件中指定要使用的默认参数值的环境名称。

默认值：`default`

有关配置文件的详细信息，请参阅 Amazon SAM CLI 配置文件。

`--config-file` *TEXT*

包含要使用的默认参数值的配置文件的路径和文件名。

默认值：当前工作目录中 `samconfig.toml`。

有关配置文件的详细信息，请参阅 Amazon SAM CLI 配置文件。

`--debug`

打开调试日志记录以打印生成的调试消息 Amazon SAM CLI 带有时间戳。

`--help, -h`

显示此消息并退出。

`--output [json|table]`

指定输出结果的格式。

默认值：`table`

`--profile` *TEXT*

从您的凭证文件中选择一个特定的个人资料以获取 Amazon 凭证。

`--region` *TEXT*

设置服务的 Amazon 区域。例如，`us-east-1`。

`--save-params`

将您在命令行中提供的参数保存到 Amazon SAM 配置文件中。

`--stack-name` *TEXT*

已部署 Amazon CloudFormation 堆栈的名称。可在应用程序的 `samconfig.toml` 文件或指定的配置文件中找到堆栈名称。

如果未指定此选项，则会显示模板中定义的本地资源。

`--template-file, --template, -t` *PATH*

Amazon SAM 模板文件。

默认值：`template.[yaml|yml|json]`

示例

以 json 格式显示名为的 Amazon CloudFormation 堆栈中已部署的资源端点的输出`test-stack`。

```
$ sam list endpoints --stack-name test-stack --output json

[
  {
    "LogicalResourceId": "HelloWorldFunction",
    "PhysicalResourceId": "sam-app-test-list-HelloWorldFunction-H85Y7yIV7ZLq",
    "CloudEndpoint": "https://zt55oi7kbljxjmcoahsj3cknwu0rposq.lambda-url.us-
east-1.on.aws/",
    "Methods": "-"
  },
```

```
  {
    "LogicalResourceId": "ServerlessRestApi",
    "PhysicalResourceId": "uj80uoe2o2",
    "CloudEndpoint": [
      "https://uj80uoe2o2.execute-api.us-east-1.amazonaws.com/Prod",
      "https://uj80uoe2o2.execute-api.us-east-1.amazonaws.com/Stage"
    ],
    "Methods": [
      "/hello['get']"
    ]
  }
]
```

sam list resources

本页提供 Amazon Serverless Application Model 命令行界面的参考信息 (Amazon SAM CLI) `sam list resources` 子命令。

有关简介 Amazon SAM CLI，请参阅 那是什么 Amazon SAMCLI？

`sam list resources`子命令显示部署时 Amazon SAM 转换在您的 Amazon Serverless Application Model (Amazon SAM) 模板中创建 Amazon CloudFormation 的资源。

在部署之前sam list resources与 Amazon SAM 模板一起使用以查看将要创建的资源。提供 Amazon CloudFormation 堆栈名称以查看包含已部署资源的合并列表。

> **Note**
>
> 要根据 Amazon SAM 模板生成资源列表，需要对模板进行本地转换。此列表中包含有条件地部署（例如，在特定区域内）的资源。

使用量

```
$ sam list resources <options>
```

选项

`--config-env` *TEXT*

在配置文件中指定要使用的默认参数值的环境名称。

默认值：`default`

有关配置文件的详细信息，请参阅 Amazon SAM CLI 配置文件。

`--config-file` *TEXT*

包含要使用的默认参数值的配置文件的路径和文件名。

默认值：当前工作目录中 `samconfig.toml`。

有关配置文件的详细信息，请参阅 Amazon SAM CLI 配置文件。

`--debug`

打开调试日志以打印生成的调试消息 Amazon SAM CLI 带有时间戳。

`--help, -h`

显示此消息并退出。

`--output [json|table]`

指定输出结果的格式。

默认值：`table`

`--profile` *TEXT*

从您的凭证文件中选择一个特定的个人资料以获取 Amazon 凭证。

`--region` *TEXT*

设置服务的 Amazon 区域。例如，`us-east-1`。

`--save-params`

将您在命令行中提供的参数保存到 Amazon SAM 配置文件中。

`--stack-name` *TEXT*

已部署 Amazon CloudFormation 堆栈的名称。可在应用程序的 `samconfig.toml` 文件或指定的配置文件中找到堆栈名称。

如果提供，则模板 IDs 中的逻辑资源将映射到其对应IDs 的物理中 Amazon CloudFormation。要了解有关物理的更多信息 IDs，请参阅Amazon CloudFormation 用户指南中的资源字段。

如果未指定此选项，则会显示模板中定义的本地资源。

`--template-file, --template, -t` *PATH*

Amazon SAM 模板文件。

默认值：`template.[yaml|yml|json]`

示例

以表格格式显示 Amazon SAM 模板中的本地资源和名为的 Amazon CloudFormation 堆栈中已部署资源的输出`test-stack`。从与本地模板相同的目录中运行。

```
$ sam list resources --stack-name test-stack --output table

-----------------------------------------------------------------------------------------------
Logical ID                                                    Physical ID
-----------------------------------------------------------------------------------------------
HelloWorldFunction                                            sam-app-test-list-
HelloWorldFunction-H85Y7yIV7ZLq
HelloWorldFunctionHelloWorldPermissionProd                    sam-app-test-list-

 HelloWorldFunctionHelloWorldPermissionProd-1QH7CPOCBL2IK
HelloWorldFunctionRole                                        sam-app-test-list-
HelloWorldFunctionRole-SRJDMJ6F7F41
ServerlessRestApi                                             uj80uoe2o2
ServerlessRestApiDeployment47fc2d5f9d                         pncw5f
ServerlessRestApiProdStage                                    Prod
ServerlessRestApiDeploymentf5716dc08b                         -
-----------------------------------------------------------------------------------------------
```

sam list stack-outputs

本页提供 Amazon Serverless Application Model 命令行界面的参考信息 (Amazon SAM CLI) `sam list stack-outputs` 子命令。

有关简介 Amazon SAM CLI，请参阅 那是什么 Amazon SAMCLI？

该`sam list stack-outputs`子命令显示来自 Amazon Serverless Application Model (Amazon SAM) 或 Amazon CloudFormation 模板的 Amazon CloudFormation 堆栈输出。有关 `Outputs` 的更多信息，请参阅《Amazon CloudFormation 用户指南》中的输出。

使用量

```
$ sam list stack-outputs <options>
```

选项

`--config-env` *TEXT*

在配置文件中指定要使用的默认参数值的环境名称。

默认值：`default`

有关配置文件的详细信息，请参阅 Amazon SAM CLI 配置文件。

`--config-file` *TEXT*

包含要使用的默认参数值的配置文件的路径和文件名。

默认值：当前工作目录中 `samconfig.toml`。

有关配置文件的详细信息，请参阅 Amazon SAM CLI 配置文件。

`--debug`

打开调试日志以打印生成的调试消息 Amazon SAM CLI 带有时间戳。

`--help, -h`

显示此消息并退出。

`--output [json|table]`

指定输出结果的格式。

默认值：`table`

`--profile` *TEXT*

从您的凭证文件中选择一个特定的个人资料以获取 Amazon 凭证。

`--region` *TEXT*

设置服务的 Amazon 区域。例如，`us-east-1`。

`--save-params`

将您在命令行中提供的参数保存到 Amazon SAM 配置文件中。

`--stack-name` *`TEXT`*

已部署 Amazon CloudFormation 堆栈的名称。可在应用程序的 `samconfig.toml` 文件或指定的配置文件中找到堆栈名称。

此选项是必需的。

示例

以表格格式显示 Amazon CloudFormation 堆栈中名为的资源的输出`test-stack`。

```
$ sam list stack-outputs --stack-name test-stack --output table

-------------------------------------------------------------------------------------------------------
OutputKey                                 OutputValue
 Description
-------------------------------------------------------------------------------------------------------
HelloWorldFunctionIamRole                 arn:aws:iam::account-number:role/sam-
 Implicit IAM Role created for Hello
                                          app-test-list-HelloWorldFunctionRole-    World
 function
                                          SRJDMJ6F7F41
HelloWorldApi                             https://uj80uoe2o2.execute-api.us-       API
 Gateway endpoint URL for Prod
                                          east-1.amazonaws.com/Prod/hello/         stage
 for Hello World function
HelloWorldFunction                        arn:aws:lambda:us-                       Hello
 World Lambda Function ARN
                                          east-1:account-number:function:sam-app-
                                          test-list-
                                          HelloWorldFunction-H85Y7yIV7ZLq
-------------------------------------------------------------------------------------------------------
```

sam local generate-event

本页提供 Amazon Serverless Application Model 命令行界面的参考信息 (Amazon SAM CLI) `sam local generate-event` 子命令。

- 有关简介 Amazon SAM CLI，请参阅 那是什么 Amazon SAMCLI？
- 有关使用方法的文档 Amazon SAM CLI `sam local generate-event`命令，请参阅使用测试简介 sam local generate-event。

`sam local generate-event` 子命令可为受支持的 Amazon Web Services 服务生成事件有效负载示例。

使用量

```
$ sam local generate-event <options> <service> <event> <event-options>
```

选项

`--config-env TEXT`

在配置文件中指定要使用的默认参数值的环境名称。默认值为“default”。有关配置文件的详细信息，请参阅 Amazon SAM CLI 配置文件。

`--config-file PATH`

包含要使用的默认参数值的配置文件的路径和文件名。在项目目录的根目录中，默认值为 `samconfig.toml`。有关配置文件的详细信息，请参阅 Amazon SAM CLI 配置文件。

`--help`

显示此消息并退出。

服务

要查看受支持服务的列表，请运行以下命令：

```
$ sam local generate-event
```

事件

要查看包含可为每项服务生成的受支持事件的列表，请运行以下命令：

```
$ sam local generate-event <service>
```

事件选项

要查看包含可修改的受支持事件选项的列表，请运行以下命令：

```
$ sam local generate-event <service> <event> --help
```

示例

有关使用 `sam local generate-event` 子命令的示例，请参阅生成示例事件。

sam local invoke

本页提供 Amazon Serverless Application Model 命令行界面的参考信息 (Amazon SAM CLI) `sam local invoke` 子命令。

- 有关简介 Amazon SAM CLI，请参阅 那是什么 Amazon SAMCLI？
- 有关使用方法的文档 Amazon SAM CLI `sam local invoke`子命令，请参阅使用测试简介 sam local invoke。

`sam local invoke`子命令在本地启动一次性调用函数。 Amazon Lambda

使用量

```
$ sam local invoke <arguments> <options>
```

> **Note**
>
> 如果 Amazon SAM 模板中定义了多个函数，请提供要调用的函数逻辑 ID。

参数

资源 ID

要调用的 Lambda 函数 的 ID。

此参数是可选的。如果您的应用程序包含单个 Lambda 函数，则 CL Amazon SAM I 将调用该函数。如果应用程序包含多个函数，请提供要调用的函数的 ID。

有效值：资源的逻辑 ID 或资源 ARN。

选项

`--add-host` *`LIST`*

将主机名到 IP 地址的映射传递到 Docker 容器的主机文件。此参数可以传递多次。

Example

示例：`--add-host` *`example.com:127.0.0.1`*

`--beta-features | --no-beta-features`

允许或拒绝测试版功能。

`--config-env` *`TEXT`*

在配置文件中指定要使用的默认参数值的环境名称。默认值为“default”。有关配置文件的详细信息，请参阅 Amazon SAM CLI 配置文件。

`--config-file` *`PATH`*

包含要使用的默认参数值的配置文件的路径和文件名。在项目目录的根目录中，默认值为“`samconfig.toml`”。有关配置文件的详细信息，请参阅 Amazon SAM CLI 配置文件。

`--container-env-vars`

（可选）在本地调试时，将环境变量传递到 Lambda 函数映像容器。

`--container-host` *`TEXT`*

本地模拟的 Lambda 容器的主机。默认值为 `localhost`。如果你想跑 Amazon SAM CLI 在 macOS 上的 Docker 容器中，你可以指定。`host.docker.internal`如果你想在与之不同的主机上运行容器 Amazon SAM CLI，您可以指定远程主机的 IP 地址。

`--container-host-interface` *`TEXT`*

应与容器端口绑定的主机网络接口的 IP 地址。默认值为 `127.0.0.1`。使用 `0.0.0.0` 可绑定到所有接口。

`--debug`

开启调试日志记录以打印调试消息 Amazon SAM CLI 生成，并显示时间戳。

`--debug-args` *`TEXT`*

要传递给调试程序的其他参数。

`--debug-port, -d` *`TEXT`*

指定后，在调试模式下启动 Lambda 函数容器，并在本地主机上开放此端口。

`--debugger-path` *`TEXT`*

挂载到 Lambda 容器中的调试程序的主机路径。

`--docker-network` *TEXT*

Lambda Docker 容器应连接到的现有 Docker 网络的名称或 ID，以及默认桥接网络。如果未指定此项，Lambda 容器将仅连接到默认的桥接 Docker 网络。

`--docker-volume-basedir, -v` *TEXT*

Amazon SAM 文件所在的基本目录的位置。如果 Docker 在远程计算机上运行，则必须在 Docker 计算机上挂载 Amazon SAM 文件所在的路径，并修改此值以匹配远程计算机。

`--env-vars, -n` *PATH*

包含 Lambda 函数环境变量值的 JSON 文件。有关环境变量文件的更多信息，请参阅环境变量文件。

`--event, -e` *PATH*

包含调用 Lambda 函数时传递给该函数的事件数据的 JSON 文件。如果不指定此选项，则不会假设任何事件。要从 `stdin` 中输入 JSON，必须传递值 '-'。有关来自不同 Amazon 服务的事件消息格式的详细信息，请参阅《Amazon Lambda 开发人员指南》中的使用其他服务。

`--force-image-build`

指定是否 Amazon SAM CLI 应该重建用于调用带图层的 Lambda 函数的镜像。

`--help`

显示此消息并退出。

`--hook-name TEXT`

用于扩展的挂钩的名称 Amazon SAM CLI 功能。

可接受的值：`terraform`。

`--invoke-image` *TEXT*

要用于本地函数调用的容器映像的 URI。默认情况下， Amazon SAM 从 Amazon ECR Public（列在 的图像存储库 Amazon SAM中）拉取容器映像。使用此选项可从其他位置提取映像。

例如 `sam local invoke MyFunction --invoke-image amazon/aws-sam-cli-emulation-image-python3.8`。

`--layer-cache-basedir` *DIRECTORY*

指定模板使用的层下载到的基目录的位置。

`--log-file, -l` *TEXT*

运行时日志将被发送到的日志文件。

`--mount-symlinks`

确保 Amazon SAM CLI 始终挂载文件中存在的符号链接以进行构建或调用。这仅适用于顶级目录中的符号链接（即直接位于函数根目录的符号链接）。默认情况下，除了需要在 NodeJS `node_modules` 中使用的`build-in-source`符号链接外，不会挂载符号链接。

`--no-event`

使用空事件调用函数。

`--no-memory-limit`

即使在 Amazon SAM 模板中配置了内存，也可以在本地调用期间移除容器中的内存限制。

`--parameter-overrides`

包含编码为键值对的 Amazon CloudFormation 参数覆盖的字符串。使用与 Amazon Command Line Interface (Amazon CLI) 相同的格式。的 Amazon SAM CLI format 是显式的键和值关键字，每个替换都用空格分隔。以下是两个示例：

- `--parameter-overrides ParameterKey=hello,ParameterValue=world`
- `--parameter-overrides ParameterKey=hello,ParameterValue=world ParameterKey=example1,ParameterValue=example2 ParameterKey=apple,ParameterValue=banana`

`--profile` *TEXT*

您的凭证文件中用于获取 Amazon 凭证的特定个人资料。

`--region` *TEXT*

要部署到的 Amazon 区域。例如，us-east-1。

`--runtime` *TEXT*

使用指定的运行时在本地调用 Lambda 函数。这将覆盖`template.yml`文件中定义的运行时间。这也允许在不修改原始函数配置的情况下使用不同的运行时测试 Lambda 函数。

`--save-params`

将您在命令行中提供的参数保存到 Amazon SAM 配置文件中。

`--shutdown`

在调用完成后模拟关闭事件，以测试关闭行为的扩展处理。

`--skip-prepare-infra`

如果没有进行任何基础架构更改，则跳过准备阶段。使用 `--hook-name` 选项。

`--skip-pull-image`

默认情况下， Amazon SAM CLI 检查 Lambda 最新的远程运行时环境，并自动更新您的本地映像以保持同步。

指定此选项可跳过向下拉最新版本的操作 Docker 您的 Lambda 运行时环境的图片。

`--template, -t` *`PATH`*

Amazon SAM 模板文件。

此选项与 `--hook-name` 不兼容。

> **Note**
>
> 如果指定此选项，则仅 Amazon SAM 加载模板及其指向的本地资源。

`--terraform-plan-file`

通往本地的相对或绝对路径 Terraform 使用时的计划文件 Amazon SAM CLI 替换为 Terraform Cloud。 此选项要求`--hook-name`将其设置为`terraform`。

示例

以下示例使用生成的事件进行本地测试，方法是使用 `s3.json` 事件在本地调用 Lambda 函数

```
$ sam local invoke --event events/s3.json S3JsonLoggerFunction
```

以下示例 `HelloWorldFunction`使用 Python 3.11 运行时测试该函数

```
$ sam local invoke --runtime python3.11 HelloWorldFunction
```

sam local start-api

本页提供 Amazon Serverless Application Model 命令行界面的参考信息 (Amazon SAM CLI) `sam local start-api` 子命令。

- 有关简介 Amazon SAM CLI，请参阅 那是什么 Amazon SAMCLI？
- 有关使用方法的文档 Amazon SAM CLI `sam local start-api`子命令，请参阅使用测试简介 sam local start-api。

`sam local start-api`子命令在本地运行您的 Amazon Lambda 函数，以便通过本地 HTTP 服务器主机进行测试。

使用量

```
$ sam local start-api <options>
```

选项

`--add-host` *`LIST`*

将主机名到 IP 地址的映射传递到 Docker 容器的主机文件。此参数可以传递多次。

Example

示例：`--add-host` *`example.com:127.0.0.1`*

`--beta-features | --no-beta-features`

允许或拒绝测试版功能。

`--config-env` *`TEXT`*

在配置文件中指定要使用的默认参数值的环境名称。默认值为“default”。有关配置文件的详细信息，请参阅 Amazon SAM CLI 配置文件。

`--config-file` *`PATH`*

包含要使用的默认参数值的配置文件的路径和文件名。在项目目录的根目录中，默认值为“samconfig.toml”。有关配置文件的详细信息，请参阅 Amazon SAM CLI 配置文件。

`--container-env-vars`

可选。在本地调试时，将环境变量传递到映像容器。

`--container-host` *TEXT*

本地模拟的 Lambda 容器的主机。默认值为 `localhost`。如果你想跑 Amazon SAM CLI 在 macOS 上的 Docker 容器中，你可以指定。`host.docker.internal`如果你想在与之不同的主机上运行容器 Amazon SAM CLI，您可以指定远程主机的 IP 地址。

`--container-host-interface` *TEXT*

应与容器端口绑定的主机网络接口的 IP 地址。默认值为 `127.0.0.1`。使用 `0.0.0.0` 可绑定到所有接口。

`--debug`

开启调试日志记录以打印生成的调试消息 Amazon SAM CLI 并显示时间戳。

`--debug-args` *TEXT*

要传递给调试程序的其他参数。

`--debug-function`

可选。指定在指定 `--warm-containers` 时应用调试选项的 Lambda 函数。此参数适用于 `--debug-port`、`--debugger-path` 和 `--debug-args`。

`--debug-port, -d` *TEXT*

指定后，在调试模式下启动 Lambda 函数容器，并在本地主机上开放此端口。

`--debugger-path` *TEXT*

将挂载到 Lambda 容器中的调试器的主机路径。

`--docker-network` *TEXT*

Lambda Docker 容器应连接到的现有 Docker 网络的名称或 ID，以及默认桥接网络。如果未指定此项，Lambda 容器将仅连接到默认的桥接 Docker 网络。

`--docker-volume-basedir, -v` *TEXT*

Amazon SAM 文件所在的基本目录的位置。如果 Docker 在远程计算机上运行，则必须在 Docker 计算机上挂载 Amazon SAM 文件所在的路径，并修改此值以匹配远程计算机。

`--env-vars, -n` *PATH*

包含 Lambda 函数环境变量值的 JSON 文件。

`--force-image-build`

指定是否 Amazon SAM CLI 应该重建用于调用带图层的函数的图像。

`--help`

显示此消息并退出。

`--hook-name` *TEXT*

用于扩展的挂钩的名称 Amazon SAM CLI 功能。

可接受的值：`terraform`。

`--host` *TEXT*

要绑定的本地主机名或 IP 地址（默认值：'127.0.0.1'）。

`--invoke-image` *TEXT*

要用于 Lambda 函数的容器映像的 URI。默认情况下，从 Amazon ECR Public 中 Amazon SAM 提取容器镜像。使用此选项可从其他位置提取映像。

您可以多次指定该选项。此选项的每个实例都可以采用字符串或键值对。如果指定字符串，则字符串就是要用于应用程序中所有函数的容器映像 URI。例如，`sam local start-api --invoke-image public.ecr.aws/sam/emu-python3.8`。如果指定键值对，则键是资源名称，值是要用于该资源的容器映像 URI。例如 `sam local start-api --invoke-image public.ecr.aws/sam/emu-python3.8 --invoke-image Function1=amazon/aws-sam-cli-emulation-image-python3.8` 。如果使用键值对，您可以为不同的资源指定不同的容器映像。

`--layer-cache-basedir` *DIRECTORY*

指定模板使用的层下载到的基目录的位置。

`--log-file, -l` *TEXT*

运行时日志将被发送到的日志文件。

`--no-memory-limit`

即使在 Amazon SAM 模板中配置了内存，也可以在本地调用期间移除容器中的内存限制。

`--parameter-overrides`

包含编码为键值对的 Amazon CloudFormation 参数覆盖的字符串。使用与 Amazon Command Line Interface (Amazon CLI) 相同的格式。的 Amazon SAM CLI format 是显式的键和值关键字，每个替换都用空格分隔。以下是两个示例：

- `--parameter-overrides ParameterKey=hello,ParameterValue=world`
- `--parameter-overrides ParameterKey=hello,ParameterValue=world ParameterKey=example1,ParameterValue=example2 ParameterKey=apple,ParameterValue=banana`

`--port, -p` *`INTEGER`*

要监听的本地端口号（默认值：'3000'）。

`--profile` *`TEXT`*

您的凭证文件中用于获取 Amazon 凭证的特定个人资料。

`--region` *`TEXT`*

要部署到的 Amazon 区域。例如，us-east-1。

`--save-params`

将您在命令行中提供的参数保存到 Amazon SAM 配置文件中。

`--shutdown`

在调用完成后模拟关闭事件，以测试关闭行为的扩展处理。

`--skip-prepare-infra`

如果没有进行任何基础架构更改，则跳过准备阶段。使用 `--hook-name` 选项。

`--skip-pull-image`

指定 CLI 是否应跳过下拉最新 Docker 映像获取 Lambda 运行时的操作。

`--ssl-cert-file` *`PATH`*

SSL 证书文件的路径（默认值：无）。使用此选项时，还必须使用 `--ssl-key-file` 选项。

`--ssl-key-file` *`PATH`*

SSL 密钥文件的路径（默认值：无）。使用此选项时，还必须使用 `--ssl-cert-file` 选项。

`--static-dir, -s` *`TEXT`*

位于此目录中的任何静态资产（例如CSS/JavaScript/HTML）文件都显示在/。

`--template, -t PATH`

Amazon SAM 模板文件。

> **Note**
> 如果指定此选项，则仅 Amazon SAM 加载模板及其指向的本地资源。

`--terraform-plan-file`

通往本地的相对或绝对路径 Terraform 使用时的计划文件 Amazon SAM CLI 替换为 Terraform Cloud。 此选项要求`--hook-name`将其设置为`terraform`。

`--warm-containers [EAGER | LAZY]`

可选。指定如何 Amazon SAM CLI 管理每个函数的容器。

有两个选项：

`EAGER`：在启动时加载所有函数的容器，且在两次调用之间保持不变。

`LAZY`：仅在首次调用每个函数时加载容器。这些容器会持续存在，以便进行其他调用。

示例

以下示例启动本地服务器，允许您通过 API 测试应用程序。要使此命令起作用，必须安装应用程序且 Docker 必须正在运行。

```
$ sam local start-api --port 3000
```

sam local start-lambda

本页提供 Amazon Serverless Application Model 命令行界面的参考信息 (Amazon SAM CLI) `sam local start-lambda` 子命令。

- 有关简介 Amazon SAM CLI，请参阅 那是什么 Amazon SAMCLI？
- 有关使用方法的文档 Amazon SAM CLI `sam local start-lambda`子命令，请参阅使用测试简介 sam local start-lambda。

`sam local start-lambda` 子命令启动要模拟 Amazon Lambda的本地端点。

使用量

```
$ sam local start-lambda <options>
```

选项

`--add-host` *`LIST`*

将主机名到 IP 地址的映射传递到 Docker 容器的主机文件。此参数可以传递多次。

Example

示例：`--add-host` *`example.com:127.0.0.1`*

`--beta-features | --no-beta-features`

允许或拒绝测试版功能。

`--config-env` *`TEXT`*

在配置文件中指定要使用的默认参数值的环境名称。默认值为“default”。有关配置文件的详细信息，请参阅 Amazon SAM CLI 配置文件。

`--config-file` *`PATH`*

包含要使用的默认参数值的配置文件的路径和文件名。在项目目录的根目录中，默认值为“samconfig.toml”。有关配置文件的详细信息，请参阅 Amazon SAM CLI 配置文件。

`--container-env-vars`

可选。在本地调试时，将环境变量传递到映像容器。

`--container-host` *`TEXT`*

本地模拟的 Lambda 容器的主机。默认值为 `localhost`。如果你想跑 Amazon SAM CLI 在 macOS 上的 Docker 容器中，你可以指定。`host.docker.internal`如果你想在与之不同的主机上运行容器 Amazon SAM CLI，您可以指定远程主机的 IP 地址。

`--container-host-interface` *`TEXT`*

应与容器端口绑定的主机网络接口的 IP 地址。默认值为 `127.0.0.1`。使用 `0.0.0.0` 可绑定到所有接口。

`--debug`

启用调试日志记录以打印生成的调试消息 Amazon SAM CLI 并显示时间戳。

`--debug-args` *TEXT*

要传递给调试程序的其他参数。

`--debug-function`

可选。指定在指定 `--warm-containers` 时应用调试选项的 Lambda 函数。此参数适用于 `--debug-port`、`--debugger-path` 和 `--debug-args`。

`--debug-port, -d` *TEXT*

指定后，在调试模式下启动 Lambda 函数容器，并在本地主机上开放此端口。

`--debugger-path` *TEXT*

挂载到 Lambda 容器中的调试器的主机路径。

`--docker-network` *TEXT*

Lambda Docker 容器应连接到的现有 Docker 网络的名称或 ID，以及默认桥接网络。如果未指定此项，Lambda 容器将仅连接到默认的桥接 Docker 网络

`--docker-volume-basedir, -v` *TEXT*

Amazon SAM 文件所在的基本目录的位置。如果 Docker 在远程计算机上运行，则必须在 Docker 计算机上挂载 Amazon SAM 文件所在的路径，并修改此值以匹配远程计算机。

`--env-vars, -n` *PATH*

包含 Lambda 函数环境变量值的 JSON 文件。

`--force-image-build`

指定是否 CLI 应该重建用于调用带图层的函数的图像。

`--help`

显示此消息并退出。

`--hook-name` *TEXT*

用于扩展的挂钩的名称 Amazon SAM CLI 功能。

可接受的值：`terraform`。

`--host TEXT`

要绑定的本地主机名或 IP 地址（默认值：'127.0.0.1'）。

`--invoke-image TEXT`

要用于本地函数调用的容器映像的 URI。默认情况下，从 Amazon ECR Public 中 Amazon SAM 提取容器镜像。使用此选项可从其他位置提取映像。

例如，`sam local start-lambda MyFunction --invoke-image amazon/aws-sam-cli-emulation-image-python3.8`。

`--layer-cache-basedir DIRECTORY`

指定模板使用的层下载到的基目录的位置。

`--log-file, -l TEXT`

运行时日志将被发送到的日志文件。

`--no-memory-limit`

即使在 Amazon SAM 模板中配置了内存，也可以在本地调用期间移除容器中的内存限制。

`--parameter-overrides`

包含编码为键值对的 Amazon CloudFormation 参数覆盖的字符串。使用与 Amazon Command Line Interface (Amazon CLI) 相同的格式。的 Amazon SAM CLI format 是显式的键和值关键字，每个替换都用空格分隔。以下是两个示例：

- `--parameter-overrides ParameterKey=hello,ParameterValue=world`
- `--parameter-overrides ParameterKey=hello,ParameterValue=world ParameterKey=example1,ParameterValue=example2 ParameterKey=apple,ParameterValue=banana`

`--port, -p INTEGER`

要监听的本地端口号（默认值：'3001'）。

`--profile TEXT`

您的凭证文件中用于获取 Amazon 凭证的特定个人资料。

`--region TEXT`

要部署到的 Amazon 区域。例如，us-east-1。

`--save-params`

将您在命令行中提供的参数保存到 Amazon SAM 配置文件中。

`--shutdown`

在调用完成后模拟关闭事件，以测试关闭行为的扩展处理。

`--skip-prepare-infra`

如果没有进行任何基础架构更改，则跳过准备阶段。使用 `--hook-name` 选项。

`--skip-pull-image`

指定是否 CLI 应该跳过为 Lambda 运行时提取最新 Docker 镜像的过程。

`--template, -t` *`PATH`*

Amazon SAM 模板文件。

> **Note**
>
> 如果指定此选项，则仅 Amazon SAM 加载模板及其指向的本地资源。此选项与 `--hook-name` 不兼容。

`--terraform-plan-file`

通往本地的相对或绝对路径 Terraform 使用时的计划文件 Amazon SAM CLI 替换为 Terraform Cloud。 此选项要求`--hook-name`将其设置为`terraform`。

`--warm-containers` *`[EAGER | LAZY]`*

可选。指定如何 Amazon SAM CLI 管理每个函数的容器。

有两个选项：

- `EAGER`：在启动时加载所有函数的容器，且在两次调用之间保持不变。
- `LAZY`：仅在首次调用每个函数时加载容器。这些容器会持续存在，以便进行其他调用。

示例

有关使用 `sam local start-lambda` 子命令的详细示例和深入演练，请参阅使用测试简介 sam local start-lambda。

sam logs

本页提供 Amazon Serverless Application Model 命令行界面的参考信息 (Amazon SAM CLI) `sam logs` 命令。

有关简介 Amazon SAM CLI，请参阅 那是什么 Amazon SAMCLI？

该`sam logs`命令会获取您的 Amazon Lambda 函数生成的日志。

使用量

```
$ sam logs <options>
```

选项

`--config-env` *TEXT*

在配置文件中指定要使用的默认参数值的环境名称。默认值为“default”。有关配置文件的详细信息，请参阅 Amazon SAM CLI 配置文件。

`--config-file` *PATH*

包含要使用的默认参数值的配置文件的路径和文件名。在项目目录的根目录中，默认值为“samconfig.toml”。有关配置文件的详细信息，请参阅 Amazon SAM CLI 配置文件。

`--cw-log-group` *LIST*

包括您指定的 CloudWatch 日志日志组中的日志。如果同时指定此选项`name`，则除了来自指定资源的日志外，还 Amazon SAM 包括来自指定日志组的日志。

`--debug`

开启调试日志记录以打印生成的调试消息 Amazon SAM CLI 并显示时间戳。

`---end-time, e` *TEXT*

获取到目前为止的日志。时间可以是相对值，例如“5 分钟前”、“明天”，也可以是格式化的时间戳，例如 “2018-01-01 10:10:10”。

`--filter` *TEXT*

允许您指定表达式以在日志事件中快速查找匹配字词、短语或值的日志。这可以是一个简单的关键词（例如，“错误”），也可以是 Amazon Logs 支持的 CloudWatch 模式。有关语法，请参阅 Amazon CloudWatch 日志文档。

`--help`

显示此消息并退出。

`--include-traces`

在日志输出中包含 X-Ray 跟踪。

`--name, -n` *TEXT*

要获取日志的资源名称。如果此资源是 Amazon CloudFormation 堆栈的一部分，则可以是 Amazon CloudFormation/Amazon SAM 模板中函数资源的逻辑 ID。再次重复该参数可以提供多个名称。如果资源位于嵌套堆栈中，则可以在名称前面加上嵌套堆栈名称以从该资源提取日志 (NestedStackLogicalId/ResourceLogicalId)。如果未给出资源名称，则将扫描给定的堆栈，并提取所有支持的资源的日志信息。如果您未指定此选项，则会 Amazon SAM 获取您指定的堆栈中所有资源的日志。支持以下资源类型：

- `AWS::Serverless::Function`
- `AWS::Lambda::Function`
- `AWS::Serverless::Api`
- `AWS::ApiGateway::RestApi`
- `AWS::Serverless::HttpApi`
- `AWS::ApiGatewayV2::Api`
- `AWS::Serverless::StateMachine`
- `AWS::StepFunctions::StateMachine`

`--output` *TEXT*

指定日志的输出格式。要打印格式化的日志，请指定 `text`。要以 JSON 格式打印日志，请指定 `json`。

`--profile` *TEXT*

您的凭证文件中用于获取 Amazon 凭证的特定个人资料。

`--region` *TEXT*

要部署到的 Amazon 区域。例如，us-east-1。

`--save-params`

将您在命令行中提供的参数保存到 Amazon SAM 配置文件中。

`--stack-name` *TEXT*

资源所 Amazon CloudFormation 属堆栈的名称。

`--start-time, -s` *TEXT*

从此时开始获取日志。时间可以是相对值，例如“5 分钟前”、“昨天”，也可以是格式化的时间戳，例如 “2018-01-01 10:10:10”。默认为“10 分钟前”。

`--tail, -t`

输出日志尾部内容。这会忽略结束时间参数，并在日志可用时持续获取。

示例

当您的函数是 Amazon CloudFormation 堆栈的一部分时，您可以在指定堆栈名称时使用该函数的逻辑 ID 来获取日志。

```
$ sam logs -n HelloWorldFunction --stack-name myStack
```

使用 -s (--start-time) 和 -e (--end-time) 选项查看特定时间范围的日志。

```
$ sam logs -n HelloWorldFunction --stack-name myStack -s '10min ago' -e '2min ago'
```

您还可以添加 `--tail` 选项以等待新日志并在它们到达时查看。

```
$ sam logs -n HelloWorldFunction --stack-name myStack --tail
```

使用 `--filter` 选项在日志事件中快速查找匹配字词、短语或值的日志：

```
$ sam logs -n HelloWorldFunction --stack-name myStack --filter "error"
```

查看子堆栈中资源的日志。

```
$ sam logs --stack-name myStack -n childStack/HelloWorldFunction
```

输出应用程序中所有受支持资源的日志尾部内容。

```
$ sam logs --stack-name sam-app --tail
```

在应用程序中获取特定 Lambda 函数和 API Gateway API 的日志。

```
$ sam logs --stack-name sam-app --name HelloWorldFunction --name HelloWorldRestApi
```

获取应用程序中所有受支持资源的日志，此外还从指定的日志组中获取日志。

```
$ sam logs --cw-log-group /aws/lambda/myfunction-123 --cw-log-group /aws/lambda/
myfunction-456
```

sam package

Amazon Serverless Application Model 命令行界面 (Amazon SAM CLI) 打包 Amazon SAM 应用程序。

此命令会创建一个`.zip`包含您的代码和依赖项的文件，并将该文件上传到亚马逊简单存储服务 (Amazon S3) Simple Service。 Amazon SAM 为存储在 Amazon S3 中的所有文件启用加密。然后，它会返回您的 Amazon SAM 模板副本，将对本地项目的引用替换为命令上传项目的 Amazon S3 位置。

默认情况下，当您使用此命令时， Amazon SAM CLI 假设您当前的工作目录是项目的根目录。的 Amazon SAM CLI 首先尝试查找使用sam build命令构建的模板文件，该文件位于`.aws-sam`子文件夹中，并命名为`template.yaml`。接下来， Amazon SAM CLI 尝试在当前工作目录`template.yml`中找到名为`template.yaml`或的模板文件。如果您指定该`--template`选项， Amazon SAM CLI的默认行为被覆盖，它将只打包该 Amazon SAM 模板及其指向的本地资源。

> **Note**
>
> sam deploy 现在隐式执行 `sam package` 的功能。您可以直接使用 sam deploy 命令打包和部署应用程序。

使用量

```
$ sam package <arguments> <options>
```

参数

资源 ID

要打包的 Lambda 函数的 ID。

此参数是可选的。如果您的应用程序包含单个 Lambda 函数，则 CL Amazon SAM I 会将其打包。如果应用程序包含多个函数，请提供用于打包单个函数的函数 ID。

有效值：资源的逻辑 ID 或资源 ARN。

选项

`--config-env` *TEXT*

在配置文件中指定要使用的默认参数值的环境名称。默认值为“default”。有关配置文件的详细信息，请参阅 Amazon SAM CLI 配置文件。

`--config-file` *PATH*

包含要使用的默认参数值的配置文件的路径和文件名。在项目目录的根目录中，默认值为“samconfig.toml”。有关配置文件的详细信息，请参阅 Amazon SAM CLI 配置文件。

`--debug`

开启调试日志记录以打印生成的调试消息 Amazon SAM CLI 并显示时间戳。

`--force-upload`

覆盖 Amazon S3 存储桶中的现有文件。指定此标志可上传构件，即使它们与 Amazon S3 存储桶中的现有构件匹配。

`--help`

显示此消息并退出。

`--image-repository` *TEXT*

此命令用于上传函数映像的 Amazon Elastic Container Registry (Amazon ECR) 的 URI。对于使用 `Image` 包类型声明的函数是必需的。

`--kms-key-id` *TEXT*

Amazon Key Management Service (Amazon KMS) 密钥的 ID，用于对 Amazon S3 存储桶中的静态项目进行加密。如果未指定此选项，则 Amazon SAM 使用 Amazon S3 托管的加密密钥。

`--metadata`

（可选）要附加到模板中引用的所有构件的元数据的映射。

`--no-progressbar`

将构件上传到 Amazon S3 时不显示进度条。

`--output-template-file` *PATH*

命令写入打包模板的文件的路径。如果不指定路径，则命令将模板写入标准输出。

`--profile` *TEXT*

您的凭证文件中用于获取 Amazon 凭证的特定个人资料。

`--region` *TEXT*

要部署到的 Amazon 区域。例如，us-east-1。

`--resolve-s3`

自动创建 Amazon S3 存储桶以用于打包。如果同时指定 `--s3-bucket` 和 `--resolve-s3` 选项，则会出现错误。

`--s3-bucket` *TEXT*

此命令用于上传构件的 Amazon S3 存储桶的名称。如果构件大于 51,200 字节，则需要 `--s3-bucket` 或 `--resolve-s3` 选项。如果同时指定 `--s3-bucket` 和 `--resolve-s3` 选项，则会出现错误。

`--s3-prefix` *TEXT*

上传到 Amazon S3 存储桶的构件名称中添加的前缀。前缀名称是 Amazon S3 存储桶的路径名称（文件夹名称）。这仅适用于以 `Zip` 包类型声明的函数。

`--save-params`

将您在命令行中提供的参数保存到 Amazon SAM 配置文件中。

`--signing-profiles` *LIST*

（可选）用于签署部署包的签名配置文件列表。此参数采用键值对列表，其中密钥是要签名的函数或层的名称，值是签名配置文件，可选的配置文件所有者用 `:` 分隔。例如，`FunctionNameToSign=SigningProfileName1 LayerNameToSign=SigningProfileName2:SigningProfileOwner`。

`--template-file, --template, -t` *PATH*

您的 Amazon SAM 模板所在的路径和文件名。

> **Note**
>
> 如果指定此选项，则仅 Amazon SAM 打包模板及其指向的本地资源。

`--use-json`

为 Amazon CloudFormation 模板输出 JSON。默认使用 YAML。

示例

以下示例为 Lambda 函数和应用程序创建和 CodeDeploy 打包工件。构件将上传到 Amazon S3 存储桶。命令的输出是名为 `package.yml` 的新文件。

```
$ sam package \
  --template-file template.yml \
  --output-template-file package.yml \
  --s3-bucket amzn-s3-demo-bucket
```

sam pipeline bootstrap

本页提供 Amazon Serverless Application Model 命令行界面的参考信息 (Amazon SAM CLI) `sam local pipeline bootstrap` 子命令。

有关简介 Amazon SAM CLI，请参阅 那是什么 Amazon SAMCLI？

该`sam pipeline bootstrap`子命令生成连接到 CI/CD 系统所需的 Amazon 基础设施资源。在运行 sam pipeline init 命令之前，必须针对管道中的每个部署阶段运行此步骤。

此子命令设置以下 Amazon 基础架构资源：

- 可选择通过以下方式配置管道权限：
  - 与 CI/CD 系统共享访问密钥 ID 和私有密钥访问凭证的管道 IAM 用户。

    > **Note**
    > 我们建议定期轮换访问密钥。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。

  - 通过 OIDC 受支持的 CI/CD 平台。有关在 Amazon SAM 管道中使用 OIDC 的介绍，请转到 如何对管道使用 OIDC 身份验证 Amazon SAM。
- 由担任的 Amazon CloudFormation 用于部署 Amazon SAM 应用程序的 Amazon CloudFormation 执行 IAM 角色。
- 用于存放 Amazon SAM 工件的 Amazon S3 存储桶。

- （可选）用于存放容器映像 Lambda 部署包的 Amazon ECR 映像存储库（如果资源为 `Image` 包类型）。

使用量

```
$ sam pipeline bootstrap <options>
```

选项

`--bitbucket-repo-uuid` *TEXT*

Bitbucket 存储库的 UUID。此选项特定于使用 Bitbucket OIDC 获取权限。

> **Note**
> 这个值可以在 https://bitbucket.org/*workspace*/*repository*/admin/addon/admin/pipelines/openid-connect 上找到

`--bucket` *TEXT*

存放工件的 Amazon S3 存储桶的 ARN。 Amazon SAM

`--cicd-provider` *TEXT*

管道的 CI/CD 平台。 Amazon SAM

`--cloudformation-execution-role` *TEXT*

部署应用程序堆栈 Amazon CloudFormation 时要担任的 IAM 角色的 ARN。仅当您希望使用自己的角色时提供。否则，该命令将创建一个新角色。

`--config-env` *TEXT*

在配置文件中指定要使用的默认参数值的环境名称。默认值为 **`default`**。有关配置文件的详细信息，请参阅 Amazon SAM CLI 配置文件。

`--config-file` *PATH*

包含要使用的默认参数值的配置文件的路径和文件名。在项目目录的根目录中，默认值为 `samconfig.toml`。有关配置文件的详细信息，请参阅 Amazon SAM CLI 配置文件。

`--confirm-changeset | --no-confirm-changeset`

提示确认资源的部署。

`--create-image-repository | --no-create-image-repository`

如果未提供 Amazon ECR 映像存储库，请指定是否要创建。Amazon ECR 存储库存放 Lambda 函数或包类型为 `Image` 的层的容器映像。默认为 `--no-create-image-repository`。

`--debug`

开启调试日志记录并打印调试消息 Amazon SAM CLI 生成，并显示时间戳。

`--deployment-branch` *TEXT*

将进行部署的分支的名称。此选项特定于使用 GitHub 操作 OIDC 获取权限。

`--github-org` *TEXT*

存储库所属的 GitHub 组织。如果不存在任何组织，请输入存储库所有者的用户名。此选项特定于使用 GitHub操作 OIDC 获取权限。

`--github-repo` *TEXT*

将进行部署的 GitHub 存储库的名称。此选项特定于使用 GitHub 操作 OIDC 获取权限。

`--gitlab-group` *TEXT*

存储库所属的 GitLab 群组。此选项特定于使用 GitLab OIDC 获取权限。

`--gitlab-project` *TEXT*

GitLab 项目名称。此选项特定于使用 GitLab OIDC 获取权限。

`--help, -h`

显示此消息并退出。

`--image-repository` *TEXT*

Amazon ECR 映像存储库的 ARN，该存储库存放 Lambda 函数或包类型为 `Image` 的层的容器映像。如果已提供，则会忽略 `--create-image-repository` 选项。如果未提供且已指定 `--create-image-repository`，则该命令将创建一个。

`--interactive | --no-interactive`

禁用引导参数的交互式提示，且如果缺少任何必需的参数，则会失败。默认值为 `--interactive`。对于此命令，`--stage` 是唯一需要的参数。

> **Note**
>
> 如果指定了 `--no-interactive` 和 `--use-oidc-provider`，则必须包含 OIDC 提供商的所有必需参数。

`--oidc-client-id` *TEXT*

为 OIDC 提供商配置的客户端 ID。

`--oidc-provider` *[github-actions | gitlab | bitbucket-pipelines]*

将用于 OIDC 权限的 CI/CD 提供商的名称。 GitLab GitHub、和 Bitbucket 均受支持。

`--oidc-provider-url` *TEXT*

OIDC 提供者的 URL。值必须以 **`https://`** 开头。

`--permissions-provider` *[oidc | iam]*

选择权限提供者来担任管道执行角色。默认值为 **`iam`**。

`--pipeline-execution-role` *TEXT*

管道用户在此阶段进行操作而要担任的 IAM 角色的 ARN。仅当您希望使用自己的角色时提供。如果未提供，则此命令将创建一个新角色。

`--pipeline-user` *TEXT*

与 CI/CD 系统共享其访问密钥 ID 和秘密访问密钥的 IAM 用户的 Amazon 资源名称（ARN）。它用于向此 IAM 用户授予访问相应 Amazon 账户的权限。如果未提供，此命令将创建 IAM 用户以及访问密钥 ID 和秘密访问密钥。

`--profile` *TEXT*

您的凭证文件中用于获取 Amazon 凭证的特定个人资料。

`--region` *TEXT*

要部署到的 Amazon 区域。例如，`us-east-1`。

`--save-params`

将您在命令行中提供的参数保存到 Amazon SAM 配置文件中。

`--stage` *TEXT*

相应部署阶段的名称。它用作创建 Amazon 的基础架构资源的后缀。

故障排除

错误：缺少必需参数

如果指定了 `--no-interactive` 和 `--use-oidc-provider`，但未提供任何必需的参数，则会显示此错误消息以及缺失参数的描述。

示例

以下示例创建了创建 CI/CD 系统所需的 Amazon 资源，它开启了调试日志并打印由 CI/CD 系统生成的调试消息 Amazon SAM CLI: 通过使用事件在本地调用 Lambda 函数，使用生成`s3.json`的事件进行本地测试

```
$ sam pipeline bootstrap --debug
```

sam pipeline init

本页提供 Amazon Serverless Application Model 命令行界面的参考信息 (Amazon SAM CLI) `sam pipeline init` 子命令。

有关简介 Amazon SAM CLI，请参阅 那是什么 Amazon SAMCLI？

该`sam pipeline init`子命令生成一个工作流配置文件，您的 CI/CD 系统可以使用该文件来部署无服务器应用程序。 Amazon SAM

在使用 sam pipeline init 之前，您必须引导管道中每个阶段的必要资源。为此，您可以运行 sam pipeline init --bootstrap 以获得安装和配置文件生成过程的指导，也可以参考之前使用 sam pipeline bootstrap 命令创建的资源。

使用量

```
$ sam pipeline init <options>
```

选项

`--bootstrap`

启用交互模式，引导用户创建必要的 Amazon 基础设施资源。

`--config-env` *TEXT*

在配置文件中指定要使用的默认参数值的环境名称。默认值为 `default`。有关配置文件的详细信息，请参阅 Amazon SAM CLI 配置文件。

`--config-file` *TEXT*

包含要使用的默认参数值的配置文件的路径和文件名。在项目根目录中，默认值为 `samconfig.toml`。有关配置文件的详细信息，请参阅 Amazon SAM CLI 配置文件。

`--debug`

开启调试日志记录以打印调试消息 Amazon SAM CLI 生成，并显示时间戳。

`--help, -h`

显示此消息并退出。

`--save-params`

将您在命令行中提供的参数保存到 Amazon SAM 配置文件中。

示例

以下示例向您展示了如何使用该`--bootstrap`选项来演练交互式模式，该模式将引导您创建必要的 Amazon 基础设施资源：

```
$ sam pipeline init --bootstrap
```

sam publish

本页提供 Amazon Serverless Application Model 命令行界面的参考信息 (Amazon SAM CLI) `sam publish` 命令。

有关简介 Amazon SAM CLI，请参阅 那是什么 Amazon SAMCLI？

该`sam publish`命令将 Amazon SAM 应用程序发布到 Amazon Serverless Application Repository。此命令采用打包的 Amazon SAM 模板并将应用程序发布到指定 Amazon 区域。

该`sam publish`命令期望 Amazon SAM 模板包含一个包含发布所需的应用程序元数据的`Metadata`部分。在 `Metadata` 部分中，`LicenseUrl` 和 `ReadmeUrl` 属性必须引用 Amazon Simple Storage Service (Amazon S3) 存储桶，而不是本地文件。有关 Amazon SAM 模板`Metadata`部分的更多信息，请参阅使用发布您的应用程序 Amazon SAM CLI。

默认情况下，`sam publish` 将创建应用程序为私有。在允许其他 Amazon 账户查看和部署您的应用程序之前，您必须将其共享。有关共享应用程序的信息，请参阅《Amazon Serverless Application Repository 开发人员指南》中的 Amazon Serverless Application Repository 基于资源的策略示例。

> **Note**
>
> 目前 `sam publish` 不支持发布本地指定的嵌套应用程序。如果您的应用程序包含嵌套应用程序，则在发布父应用程序 Amazon Serverless Application Repository 之前，必须将它们单独发布到中。

使用量

```
$ sam publish <options>
```

选项

`--config-env` *TEXT*

在配置文件中指定要使用的默认参数值的环境名称。默认值为“default”。有关配置文件的详细信息，请参阅 Amazon SAM CLI 配置文件。

`--config-file` *PATH*

包含要使用的默认参数值的配置文件的路径和文件名。在项目目录的根目录中，默认值为“`samconfig.toml`”。有关配置文件的详细信息，请参阅 Amazon SAM CLI 配置文件。

`--debug`

开启调试日志记录以打印调试消息 Amazon SAM CLI 生成，并显示时间戳。

`--help`

显示此消息并退出。

`--profile` *TEXT*

您的凭证文件中用于获取 Amazon 凭证的特定个人资料。

`--region` *TEXT*

要部署到的 Amazon 区域。例如，us-east-1。

`--save-params`

将您在命令行中提供的参数保存到 Amazon SAM 配置文件中。

`--semantic-version` *`TEXT`*

（可选）使用此选项提供应用程序的语义版本，该版本将覆盖模板文件 `Metadata` 部分中的 `SemanticVersion` 属性。有关语义版本控制的更多信息，请参阅语义版本控制规范。

`--template, -t` *`PATH`*

Amazon SAM 模板文件的路径`[default: template.[yaml|yml]]`。

示例

要发布应用程序，请执行以下操作：

```
$ sam publish --template packaged.yaml --region us-east-1
```

sam remote invoke

本页提供 Amazon Serverless Application Model 命令行界面的参考信息 (Amazon SAM CLI) `sam remote invoke` 命令。

- 有关简介 Amazon SAM CLI，请参阅 那是什么 Amazon SAMCLI？
- 有关使用的文档 Amazon SAM CLI `sam remote invoke`命令，请参阅云端测试简介 sam remote invoke。

`sam remote invoke` 命令调用 Amazon Web Services 云中支持的资源。

使用量

```
$ sam remote invoke <arguments> <options>
```

参数

资源 ID

要调用的支持资源的 ID。

此参数接受以下值：

- Amazon 资源名称（ARN）– 资源的 ARN。

> **Tip**
>
> 使用 `sam list stack-outputs --stack-name <stack-name>` 获取资源的 ARN。

- 逻辑 ID – 资源的逻辑 ID。您还必须使用`--stack-name`选项提供 Amazon CloudFormation 堆栈名称。
- 物理 ID – 资源的物理 ID。此 ID 是在您使用部署资源时创建的 Amazon CloudFormation。

> **Tip**
>
> 使用 `sam list resources --stack-name <stack-name>` 获取资源的物理 ID。

当您提供 ARN 或物理 ID 时：

如果您提供 ARN 或物理 ID，请不要提供堆栈名称。当使用`--stack-name`选项提供堆栈名称时，或者当堆栈名称是在配置文件中定义时， Amazon SAM CLI 会自动将您的资源 ID 作为 Amazon CloudFormation 堆栈中的逻辑 ID 值进行处理。

当您不提供资源 ID 时：

如果您没有提供资源 ID，但确实提供了带有`--stack-name`选项的堆栈名称，则 Amazon SAM CLI 将尝试使用以下逻辑自动调用 Amazon CloudFormation 堆栈中的资源：

1. 的 Amazon SAM CLI 将按以下顺序识别资源类型，并在堆栈中找到资源类型后进入下一步：
   a. Lambda
   b. Step Functions
   c. Amazon SQS
   d. Kinesis Data Streams
2. 如果资源类型的堆栈中只有一个资源，则 Amazon SAM CLI 会调用它。如果您的堆栈中存在多个该资源类型的资源，则 Amazon SAM CLI 将返回错误。

以下是一些示例 Amazon SAM CLI 会做的：

- 包含两个 Lambda 函数和一个 Amazon SQS 队列的堆栈 — Amazon SAM CLI 将找到 Lambda 资源类型并返回错误，因为堆栈包含多个 Lambda 函数。

- 包含一个 Lambda 函数和两个 Amazon Kinesis Data Streams 应用程序的堆栈 — Amazon SAM CLI 将找到 Lambda 函数并将其调用，因为堆栈包含单个 Lambda 资源。
- 包含单个 Amazon SQS 队列和两个 Kinesis Data Streams 应用程序的堆栈 — Amazon SAM CLI 将找到 Amazon SQS 队列并将其调用，因为该堆栈包含单个 Amazon SQS 队列。

选项

`--beta-features | --no-beta-features`

允许或拒绝测试版功能。

`--config-env` *`TEXT`*

从您的环境中指定要使用的环境 Amazon SAM CLI 配置文件。

默认值：`default`

`--config-file` *`FILENAME`*

指定配置文件的路径和文件名。

有关配置文件的详细信息，请参阅 正在配置 Amazon SAM CLI。

默认：`samconfig.toml` 位于项目目录的根目录。

`--debug`

启用调试登入 这将打印由生成的调试消息和时间戳 Amazon SAM CLI.

`--event, -e` *`TEXT`*

要发送到目标资源的事件。

`--event-file` *`FILENAME`*

包含要发送到目标资源的事件的文件路径。

`--help, -h`

显示帮助消息并退出。

`--output` *`[ text | json ]`*

以特定的输出格式输出调用结果。

`json` – 请求元数据和资源响应以 JSON 结构返回。响应包含完整的 SDK 输出。

`text` – 请求元数据以文本结构返回。资源响应以被调用资源的输出格式返回。

`--parameter`

其他 Boto3可以传递给正在调用的资源的参数。

Amazon Kinesis Data Streams

以下附加参数可用于将记录放入 Kinesis 数据流：

- `ExplicitHashKey='string'`
- `PartitionKey='string'`
- `SequenceNumberForOrdering='string'`
- `StreamARN='string'`

有关每个参数的描述，请参阅 Kinesis.Client.put_record。

Amazon Lambda

以下附加参数可用于调用 Lambda 资源并接收缓冲响应：

- `ClientContext='base64-encoded string'`
- `InvocationType='[ DryRun | Event | RequestResponse ]'`
- `LogType='[ None | Tail ]'`
- `Qualifier='string'`

以下附加参数可用于通过响应流式处理调用 Lambda 资源：

- `ClientContext='base64-encoded string'`
- `InvocationType='[ DryRun | RequestResponse ]'`
- `LogType='[ None | Tail ]'`
- `Qualifier='string'`

有关每个参数的说明，请参阅以下内容：

- 带有缓冲响应的 Lambda – Lambda.Client.invoke
- 带响应流式处理的 Lambda – Lambda.Client.invoke_with_response_stream

Amazon Simple Queue Service(Amazon SQS)

以下附加参数可用于向 Amazon SQS 队列发送消息：

- `DelaySeconds=integer`
- `MessageAttributes='json string'`
- `MessageDeduplicationId='string'`
- `MessageGroupId='string'`
- `MessageSystemAttributes='json string'`

有关每个参数的描述，请参阅 SQS.Client.send_message。

Amazon Step Functions

以下附加参数可用于启动状态机执行：

- `name='string'`
- `traceHeader='string'`

有关每个参数的描述，请参阅 SFN.Client.start_execution。

`--profile TEXT`

从您的凭证文件中获取 Amazon 凭证的特定个人资料。

`--region TEXT`

Amazon Web Services 区域 资源的。例如，`us-east-1`。

`--stack-name TEXT`

资源所属 Amazon CloudFormation 堆栈的名称。

`--test-event-name NAME`

要传递给 Lambda 函数的可共享测试事件的名称。

> **Note**
> 此选项仅支持 Lambda 函数。

示例

以下示例调用云中支持的资源并激活调试日志记录，该日志会打印由 Amazon 云端生成的调试消息和时间戳 Amazon SAM CLI:

```
$ sam remote invoke--debug
```

sam remote test-event

本页提供 Amazon Serverless Application Model 命令行界面的参考信息 (Amazon SAM CLI) `sam remote test-event` 命令。

- 有关简介 Amazon SAM CLI，请参阅 那是什么 Amazon SAMCLI？
- 有关使用的文档 Amazon SAM CLI `sam remote test-event`命令，请参阅云测试简介 sam remote test-event。

该`sam remote test-event`命令与 Amazon EventBridge 架构注册表中的可共享测试事件进行交互。

使用量

```
$ sam remote test-event <options> <subcommand>
```

选项

`--help, -h`

显示帮助消息并退出。

子命令

`delete`

从 EventBridge 架构注册表中删除可共享的测试事件。有关更多参考信息，请参阅 sam remote test-event delete。

`get`

从 EventBridge 架构注册表中获取可共享的测试事件。有关更多参考信息，请参阅 sam remote test-event get。

`list`

列出 Amazon Lambda 函数的现有可共享测试事件。有关更多参考信息，请参阅 sam remote test-event list。

`put`

将事件从本地文件保存到 EventBridge 架构注册表。有关更多参考信息，请参阅 sam remote test-event put。

sam remote test-event delete

本页提供 Amazon Serverless Application Model 命令行界面的参考信息 (Amazon SAM CLI) `sam remote test-event delete` 子命令。

- 有关简介 Amazon SAM CLI，请参阅 那是什么 Amazon SAMCLI？
- 有关使用方法的文档 Amazon SAM CLI `sam remote test-event`命令，请参阅云测试简介 sam remote test-event。

该`sam remote test-event delete`子命令从 Amazon EventBridge 架构注册表中删除可共享的测试事件。

使用量

```
$ sam remote test-event delete <arguments> <options>
```

参数

资源 ID

与可共享测试事件关联的 Amazon Lambda 函数的 ID。

如果您提供逻辑 ID，则还必须使用选项为与 Lambda 函数关联的 Amazon CloudFormation 堆栈提供一个值。`--stack-name`

有效值：资源的逻辑 ID 或资源 ARN.

选项

`--config-env` *`TEXT`*

在配置文件中指定要使用的默认参数值的环境名称。默认值为“default”。有关配置文件的详细信息，请参阅 Amazon SAM CLI 配置文件。

`--config-file` *`PATH`*

包含要使用的默认参数值的配置文件的路径和文件名。在项目目录的根目录中，默认值为“`samconfig.toml`”。有关配置文件的详细信息，请参阅 Amazon SAM CLI 配置文件。

`--help, -h`

显示帮助消息并退出。

`--name` *`TEXT`*

要删除的可共享测试事件的名称。

`--stack-name` *`TEXT`*

与 Lambda 函数关联的 Amazon CloudFormation 堆栈的名称。

如果您提供 Lambda 函数逻辑 ID 作为参数，则此选项为必需。

sam remote test-event get

本页提供 Amazon Serverless Application Model 命令行界面的参考信息 (Amazon SAM CLI) `sam remote test-event get` 子命令。

- 有关简介 Amazon SAM CLI，请参阅 那是什么 Amazon SAMCLI？
- 有关使用方法的文档 Amazon SAM CLI `sam remote test-event`命令，请参阅云测试简介 sam remote test-event。

该`sam remote test-event get`子命令从 Amazon EventBridge 架构注册表中获取一个可共享的测试事件。

使用量

```
$ sam remote test-event get <arguments> <options>
```

参数

资源 ID

与要获取的可共享测试事件关联的 Amazon Lambda 函数的 ID。

如果您提供逻辑 ID，则还必须使用选项为与 Lambda 函数关联的 Amazon CloudFormation 堆栈提供一个值。`--stack-name`

有效值：资源的逻辑 ID 或资源 ARN.

选项

`--config-env` *TEXT*

在配置文件中指定要使用的默认参数值的环境名称。默认值为“default”。有关配置文件的详细信息，请参阅 Amazon SAM CLI 配置文件。

`--config-file` *PATH*

包含要使用的默认参数值的配置文件的路径和文件名。在项目目录的根目录中，默认值为“`samconfig.toml`”。有关配置文件的详细信息，请参阅 Amazon SAM CLI 配置文件。

`--help, -h`

显示帮助消息并退出。

`--name` *TEXT*

要获取的可共享测试事件的名称。

`--output-file` *FILENAME*

在本地计算机上保存事件的文件路径和名称。

如果您不提供此选项，则 Amazon SAM CLI 会将可共享测试事件的内容输出到您的主机。

`--stack-name` *TEXT*

与 Lambda 函数关联的 Amazon CloudFormation 堆栈的名称。

如果您提供 Lambda 函数逻辑 ID 作为参数，则此选项为必需。

sam remote test-event list

本页提供 Amazon Serverless Application Model 命令行界面的参考信息 (Amazon SAM CLI) `sam remote test-event list` 子命令。

- 有关简介 Amazon SAM CLI，请参阅 那是什么 Amazon SAMCLI？
- 有关使用方法的文档 Amazon SAM CLI `sam remote test-event`命令，请参阅云测试简介 sam remote test-event。

该`sam remote test-event list`子命令列出了 Amazon EventBridge 架构注册表中特定 Amazon Lambda 函数的现有可共享测试事件。

使用量

```
$ sam remote test-event list <arguments> <options>
```

参数

资源 ID

与可共享测试事件关联的 Lambda 函数的 ID。

如果您提供逻辑 ID，则还必须使用选项为与 Lambda 函数关联的 Amazon CloudFormation 堆栈提供一个值。`--stack-name`

有效值：资源的逻辑 ID 或资源 ARN.

选项

`--config-env` *TEXT*

在配置文件中指定要使用的默认参数值的环境名称。默认值为“default”。有关配置文件的详细信息，请参阅 Amazon SAM CLI 配置文件。

`--config-file` *PATH*

包含要使用的默认参数值的配置文件的路径和文件名。在项目目录的根目录中，默认值为“`samconfig.toml`”。有关配置文件的详细信息，请参阅 Amazon SAM CLI 配置文件。

`--help, -h`

显示帮助消息并退出。

`--stack-name` *TEXT*

与 Lambda 函数关联的 Amazon CloudFormation 堆栈的名称。

如果您提供 Lambda 函数逻辑 ID 作为参数，则此选项为必需。

示例

有关使用此命令的示例，请参阅列出可共享测试事件。

sam remote test-event put

本页提供 Amazon Serverless Application Model 命令行界面的参考信息 (Amazon SAM CLI) `sam remote test-event put` 子命令。

- 有关简介 Amazon SAM CLI，请参阅 那是什么 Amazon SAMCLI？
- 有关使用方法的文档 Amazon SAM CLI `sam remote test-event`命令，请参阅云测试简介 sam remote test-event。

该`sam remote test-event put`子命令将本地计算机上的可共享测试事件保存到 Amazon EventBridge 架构注册表中。

使用量

```
$ sam remote test-event put <arguments> <options>
```

参数

资源 ID

与可共享测试事件关联的 Amazon Lambda 函数的 ID。

如果您提供逻辑 ID，则还必须使用选项为与 Lambda 函数关联的 Amazon CloudFormation 堆栈提供一个值。`--stack-name`

有效值：资源的逻辑 ID 或资源 ARN.

选项

`--config-env` *TEXT*

在配置文件中指定要使用的默认参数值的环境名称。默认值为“default”。有关配置文件的详细信息，请参阅 Amazon SAM CLI 配置文件。

`--config-file` *PATH*

包含要使用的默认参数值的配置文件的路径和文件名。在项目目录的根目录中，默认值为“`samconfig.toml`”。有关配置文件的详细信息，请参阅 Amazon SAM CLI 配置文件。

`--file` *FILENAME*

本地计算机上事件的文件路径和名称。

提供 - 作为要从 stdin 处读取的文件名值。

此选项是必需的。

`--force, -f`

覆盖同名的可共享测试事件。

`--help, -h`

显示帮助消息并退出。

`--name` *TEXT*

用于保存可共享测试事件的名称。

如果 EventBridge 架构注册表中存在同名的可共享测试事件，则 Amazon SAM CLI 不会覆盖它。若要覆盖，请添加 `--force` 选项。

`--output-file` *FILENAME*

在本地计算机上保存事件的文件路径和名称。

如果您不提供此选项， Amazon SAM CLI 会将可共享测试事件的内容输出到您的主机。

`--stack-name` *TEXT*

与 Lambda 函数关联的 Amazon CloudFormation 堆栈的名称。

如果您提供 Lambda 函数逻辑 ID 作为参数，则此选项为必需。

示例

有关使用此命令的示例，请参阅保存可共享测试事件。

sam sync

本页提供 Amazon Serverless Application Model 命令行界面的参考信息 (Amazon SAM CLI) `sam sync` 命令。

- 有关简介 Amazon SAM CLI，请参阅 那是什么 Amazon SAMCLI？
- 有关使用的文档 Amazon SAM CLI，请参阅 的 Amazon SAM CLI。

`sam sync` 命令将本地应用程序的更改同步到 Amazon Web Services 云。

使用量

```
$ sam sync <options>
```

选项

`--base-dir, -s DIRECTORY`

根据此目录解析函数或层的源代码的相对路径。使用此选项更改源代码文件夹相对路径的解析方式。默认情况下，相对路径是根据 Amazon SAM 模板的位置进行解析的。

除了正在构建的根应用程序或堆栈中的资源外，此选项还适用于嵌套应用程序或堆栈。此外，此选项适用于以下资源类型和属性：

- 资源类型：`AWS::Serverless::Function` 属性：`CodeUri`
- 资源类型：`AWS::Serverless::Function` 资源属性：`Metadata` 条目：`DockerContext`
- 资源类型：`AWS::Serverless::LayerVersion` 属性：`ContentUri`
- 资源类型：`AWS::Lambda::Function` 属性：`Code`
- 资源类型：`AWS::Lambda::LayerVersion` 属性：`Content`

`--build-image TEXT`

构建应用程序时要使用的容器映像的 URI。默认情况下， Amazon SAM 使用亚马逊弹性容器注册表 (Amazon ECR) Public 中的容器镜像存储库 URI。指定此选项以使用不同的映像。

您可以在单个命令中多次使用此选项。每种选项都接受一个字符串或一个键值对。

- 字符串 – 指定应用程序中所有资源都将使用的容器映像的 URI。以下是示例：

  ```
  $ sam sync --build-image amazon/aws-sam-cli-build-image-python3.8
  ```

- 键值对 – 将资源名称指定为键，并将要与该资源一起使用的容器映像 URI 指定为值。使用此格式为应用程序中的每个资源指定不同的容器映像 URI。以下是示例：

  ```
  $ sam sync --build-image Function1=amazon/aws-sam-cli-build-image-python3.8
  ```

在指定了 `--use-container` 选项的情况下此选项才适用，否则会导致错误。

`--build-in-source | --no-build-in-source`

提供 `--build-in-source` 以便直接在源文件夹中生成项目。

`--build-in-source` 选项支持以下运行时和构建方法：

- 运行时 — 任意 Node.js 该`sam init --runtime`选项支持的运行时间。
- 构建方法：`Makefile`、`esbuild`。

`--build-in-source` 选项与以下选项不兼容：

- `--use-container`

默认值：`--no-build-in-source`

`--capabilities` *`LIST`*

您指定允许创建特定堆栈 Amazon CloudFormation 的功能列表。某些堆栈模板可能包含可能会影响您的权限的资源 Amazon Web Services 账户。例如，通过创建新 Amazon Identity and Access Management (IAM) 用户。指定此选项以覆盖默认值。有效值包括：

- CAPABILITY_IAM
- CAPABILITY_NAMED_IAM
- CAPABILITY_RESOURCE_POLICY
- CAPABILITY_AUTO_EXPAND

默认值：`CAPABILITY_NAMED_IAM` 和 `CAPABILITY_AUTO_EXPAND`

`--code`

默认情况下， Amazon SAM 同步应用程序中的所有资源。指定此选项可仅同步代码资源，包括以下内容：

- `AWS::Serverless::Function`
- `AWS::Lambda::Function`
- `AWS::Serverless::LayerVersion`
- `AWS::Lambda::LayerVersion`
- `AWS::Serverless::Api`
- `AWS::ApiGateway::RestApi`
- `AWS::Serverless::HttpApi`
- `AWS::ApiGatewayV2::Api`
- `AWS::Serverless::StateMachine`
- `AWS::StepFunctions::StateMachine`

要同步代码资源，请 APIs 直接 Amazon SAM 使用 Amazon 服务，而不是通过部署 Amazon CloudFormation。要更新您的 Amazon CloudFormation 堆栈，请运行sam sync --watch或sam deploy。

`--config-env` *TEXT*

在配置文件中指定要使用的默认参数值的环境名称。默认值为“default”。有关配置文件的详细信息，请参阅 Amazon SAM CLI 配置文件。

`--config-file` *PATH*

包含要使用的默认参数值的配置文件的路径和文件名。在项目目录的根目录中，默认值为“`samconfig.toml`”。有关配置文件的详细信息，请参阅 Amazon SAM CLI 配置文件。

`--dependency-layer | --no-dependency-layer`

指定是否将单个函数的依赖项分离到另一层以加快同步过程。

默认值：`--dependency-layer`

`--image-repository` *TEXT*

此命令用于上传函数映像的 Amazon Elastic Container Registry (Amazon ECR) 存储库的名称。对于使用 `Image` 包类型声明的函数是必需的。

`--image-repositories` *TEXT*

函数与其 Amazon ECR 存储库 URI 之间的映射。通过逻辑 ID 引用函数。以下是示例：

```
$ sam sync --image-repositories Function1=123456789012.dkr.ecr.us-
east-1.amazonaws.com/my-repo
```

您可以在单个命令中多次指定此选项。

`--kms-key-id` *TEXT*

Amazon Key Management Service (Amazon KMS) 密钥的 ID，用于对 Amazon S3 存储桶中的静态项目进行加密。如果您未指定此选项，则 Amazon SAM 使用 Amazon S3 托管的加密密钥。

`--metadata`

要附加到模板中引用的所有构件的元数据的映射。

`--notification-arns` *LIST*

与堆栈关联的亚马逊简单通知服务 (Amazon SNS) Service ARNs 主题 Amazon CloudFormation 列表。

`--no-use-container`

允许您使用 IDE 工具包设置默认行为的选项。

`--parameter-overrides`

包含编码为键值对的 Amazon CloudFormation 参数覆盖的字符串。使用与 Amazon Command Line Interface (Amazon CLI) 相同的格式。的 Amazon SAM CLI format 是显式的键和值关键字，每个替换都用空格分隔。以下是两个示例：

- `--parameter-overrides ParameterKey=hello,ParameterValue=world`
- `--parameter-overrides ParameterKey=hello,ParameterValue=world ParameterKey=example1,ParameterValue=example2 ParameterKey=apple,ParameterValue=banana`

`--resource` *TEXT*

指定要同步的资源类型。要同步多个资源，可以多次指定此选项。`--code` 选项支持此选项。该值必须是 `--code` 下列出的资源之一。例如，`--resource AWS::Serverless::Function --resource AWS::Serverless::LayerVersion`。

`--resource-id` *TEXT*

指定要同步的资源 ID。要同步多个资源，可以多次指定此选项。`--code` 选项支持此选项。例如，`--resource-id Function1 --resource-id Function2`。

`--role-arn` *TEXT*

应用变更集时 Amazon CloudFormation 扮演的 IAM 角色的 Amazon 资源名称 (ARN)。

`--s3-bucket` *TEXT*

此命令用于上传 Amazon CloudFormation 模板的亚马逊简单存储服务 (Amazon S3) 存储桶的名称。如果模板大于 51,200 字节，则需要 `--s3-bucket` 或 `--resolve-s3` 选项。如果同时指定 `--s3-bucket` 和 `--resolve-s3` 选项，则会出现错误。

`--s3-prefix` *TEXT*

上传到 Amazon S3 存储桶的构件的名称中添加的前缀。前缀名称是 Amazon S3 存储桶的路径名称（文件夹名称）。这仅适用于使用 `Zip` 包类型声明的函数。

`--save-params`

将您在命令行中提供的参数保存到 Amazon SAM 配置文件中。

`--skip-deploy-sync | --no-skip-deploy-sync`

若不需要，则指定 `--skip-deploy-sync` 以跳过初始基础设施同步。这些区域有： Amazon SAM CLI 会将您的本地 Amazon SAM 模板与已部署的 Amazon CloudFormation 模板进行比较，并且仅在检测到更改时才执行部署。

指定`--no-skip-deploy-sync`每次运行时`sam sync`都执行 Amazon CloudFormation 部署。

要了解更多信息，请参阅 跳过初始 Amazon CloudFormation 部署。

默认值：`--skip-deploy-sync`

`--stack-name` *TEXT*

您的应用程序的 Amazon CloudFormation 堆栈名称。

此选项是必需的。

`--tags` *LIST*

要与已创建或更新的堆栈关联的标签列表。 Amazon CloudFormation 还会将这些标签传播到堆栈中支持它的资源。

`--template-file, --template, -t` *PATH*

Amazon SAM 模板所在的路径和文件名。

> **Note**
>
> 如果指定此选项，则仅 Amazon SAM 部署模板及其指向的本地资源。

`--use-container, -u`

如果您的函数依赖于具有本机编译依赖关系的包，请使用此选项在-like 中构建函数 Amazon LambdaDocker 容器。

> **Note**
>
> 目前，此选项与 `--dependency-layer` 不兼容。如果`--use-container`与一起使用`--dependency-layer`，则 Amazon SAM CLI 通知你并继续。`--no-dependency-layer`

`--watch`

启动一个进程，监视您的本地应用程序是否有更改，并自动将其同步到。 Amazon Web Services 云默认情况下，当您指定此选项时，会在更新应用程序中的所有资源时 Amazon SAM 同步它们。使用此选项， Amazon SAM 执行初始 Amazon CloudFormation 部署。然后， Amazon SAM APIs 使用 Amazon 服务更新代码资源。 Amazon SAM Amazon CloudFormation 用于在更新 Amazon SAM 模板时更新基础架构资源。

`--watch-exclude` *`TEXT`*

禁止文件或文件夹用于文件更改的观测。如果要使用此选项，则还必须提供 `--watch`。

此选项收到键值对：

- 键：应用程序中 Lambda 函数的逻辑 ID。
- 值：要排除的关联文件名或文件夹。

更新使用该`--watch-exclude`选项指定的任何文件或文件夹时， Amazon SAM CLI 不会启动同步。但是，当对其他文件或文件夹的更新启动同步时，这些文件或文件夹将包含在该同步中。

您可以在单个命令中多次提供此选项。

示例

有关使用此命令的示例，请参阅sam sync 命令的选项。

sam traces

本页提供 Amazon Serverless Application Model 命令行界面的参考信息 (Amazon SAM CLI) `sam traces` 命令。

有关简介 Amazon SAM CLI，请参阅 那是什么 Amazon SAMCLI？

该`sam traces`命令将在中获取您的 Amazon X-Ray Amazon Web Services 账户 痕迹。 Amazon Web Services 区域

使用量

```
$ sam traces <options>
```

选项

`--config-env` *TEXT*

在配置文件中指定要使用的默认参数值的环境名称。默认值为“default”。有关配置文件的详细信息，请参阅 Amazon SAM CLI 配置文件。

`--config-file` *PATH*

包含要使用的默认参数值的配置文件的路径和文件名。在项目目录的根目录中，默认值为“`samconfig.toml`”。有关配置文件的详细信息，请参阅 Amazon SAM CLI 配置文件。

`--end-time` *TEXT*

获取到此时为止的跟踪。时间可以是相对值，例如“5 分钟前”、“明天”，也可以是格式化的时间戳，例如 “2018-01-01 10:10:10”。

`--output` *TEXT*

指定日志的输出格式。要打印格式化的日志，请指定 `text`。要以 JSON 格式打印日志，请指定 `json`。

`--save-params`

将您在命令行中提供的参数保存到 Amazon SAM 配置文件中。

`--start-time` *TEXT*

获取从此时开始的跟踪。时间可以是相对值，例如“5 分钟前”、“昨天”，也可以是格式化的时间戳，例如 “2018-01-01 10:10:10”。默认为“10 分钟前”。

`--tail`

跟进跟踪输出。这将忽略结束时间参数，并在跟踪可用时持续显示跟踪。

`--trace-id` *TEXT*

X-Ray 跟踪的唯一标识符。

示例

运行以下命令按 ID 获取 X-Ray 跟踪。

```
$ sam traces --trace-id tracing-id-1 --trace-id tracing-id-2
```

运行以下命令在 X-Ray 跟踪可用时对其进行跟进。

```
$ sam traces --tail
```

sam validate

本页提供 Amazon Serverless Application Model 命令行界面的参考信息 (Amazon SAM CLI) `sam validate` 命令。

有关简介 Amazon SAM CLI，请参阅 那是什么 Amazon SAMCLI？

该`sam validate`命令验证 Amazon SAM 模板文件是否有效。

使用量

```
$ sam validate <options>
```

选项

`--config-env` *TEXT*

在配置文件中指定要使用的默认参数值的环境名称。默认值为“default”。有关配置文件的详细信息，请参阅 Amazon SAM CLI 配置文件。

`--config-file` *PATH*

包含要使用的默认参数值的配置文件的路径和文件名。在项目目录的根目录中，默认值为“samconfig.toml”。有关配置文件的详细信息，请参阅 Amazon SAM CLI 配置文件。

`--debug`

启用调试日志记录以打印生成的调试消息 Amazon SAM CLI 并显示时间戳。

`--lint`

通过 cfn-lint 对模板运行检查验证。创建 `cfnlintrc` 配置文件以指定其他参数。有关更多信息，请参阅存储库中的 cfn-lint。Amazon CloudFormation GitHub

`--profile` *TEXT*

您的凭证文件中用于获取 Amazon 凭证的特定个人资料。

`--region` *TEXT*

要部署到的 Amazon 区域。例如，us-east-1。

`--save-params`

将您在命令行中提供的参数保存到 Amazon SAM 配置文件中。

`--template-file, --template, -t` *PATH*

Amazon SAM 模板文件。默认值为 `template.[yaml|yml]`。

如果您的模板位于当前工作目录中且命名为 `template.[yaml|yml|json]`，则不需要此选项。

如果您刚刚运行 sam build，则不需要此选项。

示例

有关使用此命令验证模板的示例，请参阅验证 Amazon SAM 模板文件。

有关将此命令与 cfn-lint 一起使用的示例，请参阅使用 Amazon CloudFormation Linter 验证您的 Amazon SAM 应用程序。

Amazon SAM CLI management

本节包含有关如何管理和自定义版本的信息 Amazon SAM CLI。 这包括有关如何配置的信息 Amazon SAM CLI 使用项目级配置文件的命令参数值。它还包括有关管理您的不同版本的信息 Amazon SAM CLI，设置 Amazon 凭据以便 Amazon SAM 可以代表您拨打 Amazon 服务电话，以及您可以自定义的不同方式 Amazon SAM。本节最后有一节介绍一般 Amazon SAM 故障排除。

主题

- Amazon SAM CLI 配置文件
- 管理 Amazon SAM CLI 版本
- 设置 Amazon 凭证
- 遥测中的应用 Amazon SAM CLI
- Amazon SAM CLI 故障排除

Amazon SAM CLI 配置文件

Amazon Serverless Application Model 命令行界面 (Amazon SAM CLI) 支持可用于配置的项目级配置文件 Amazon SAM CLI 命令参数值。

有关创建和使用配置文件的文档，请参阅正在配置 Amazon SAM CLI。

主题

- 默认配置文件设置
- 支持的配置文件格式
- 指定配置文件
- 配置文件基础
- 参数值规则
- 配置优先级
- 创建和修改配置文件

默认配置文件设置

Amazon SAM 使用以下默认配置文件设置：

- 名称 – `samconfig`。
- 位置 - 位于项目的根目录中。此位置与 `template.yaml` 文件的位置相同。
- 格式 – TOML。要了解更多信息，请参阅中的 TOML TOML 文档。

以下是一个包含默认配置文件名和位置的示例项目结构：

```
sam-app
### README.md
### __init__.py
### events
### hello_world
### samconfig.toml
### template.yaml
### tests
```

以下是 `samconfig.toml` 文件示例：

```
...
version = 0.1

[default]
[default.global]
```

```
[default.global.parameters]
stack_name = "sam-app"

[default.build.parameters]
cached = true
parallel = true

[default.deploy.parameters]
capabilities = "CAPABILITY_IAM"
confirm_changeset = true
resolve_s3 = true

[default.sync.parameters]
watch = true

[default.local_start_api.parameters]
warm_containers = "EAGER"

[prod]
[prod.sync]
[prod.sync.parameters]
watch = false
```

支持的配置文件格式

支持 TOML 和 [YAML|YML] 格式。请参阅以下基本语法：

TOML

```
version = 0.1
[environment]
[environment.command]
[environment.command.parameters]
option = parameter value
```

YAML

```
version: 0.1
environment:
  command:
    parameters:
      option: parameter value
```

指定配置文件

默认情况下， Amazon SAM CLI 按以下顺序查找配置文件：

1. 自定义配置文件-如果使用`--config-file`选项来指定文件名和位置，则 Amazon SAM CLI 先查找此文件。
2. 默认 **`samconfig.toml`** 文件 - 这是默认配置文件名和格式，位于项目的根目录中。如果您未指定自定义配置文件，则 Amazon SAM CLI 接下来要寻找这个文件。
3. **`samconfig.[yaml|yml]`**file — 如果项目根目录中`samconfig.toml`不存在，则 Amazon SAM CLI 正在寻找这个文件。

以下示例说明了如何使用 `--config-file` 选项指定自定义配置文件：

```
$ sam deploy --config-file myconfig.yaml
```

> **Note**
>
> 该`--config-file`参数必须相对于 Amazon SAM 模板文件的位置，因为 Amazon SAM CLI 需要确定应用配置的上下文。该`samconfig.toml`文件管理您的版本的配置设置 Amazon SAM CLI，然后 CLI 在`samconfig.toml`文件的相对文件夹中查找文件（或覆盖的配置文件参数）。`template.yaml`

配置文件基础

环境

环境是包含一组唯一配置设置的命名标识符。您可以在一个 Amazon SAM 应用程序中拥有多个环境。

默认环境名称是 `default`。

使用 Amazon SAM CLI `--config-env`选项来指定要使用的环境。

命令

命令是 Amazon SAM CLI 命令为其指定参数值。

要为所有命令指定参数值，请使用 `global` 标识符。

当引用 Amazon SAM CLI 命令，将空格 ( ) 和连字符 () 替换为下划线 (–_)。请参阅以下示例：

- `build`
- `local_invoke`
- `local_start_api`

参数

参数指定为键值对。

- 关键是 Amazon SAM CLI 命令选项名称。
- 值是要指定的值。

指定键时，请使用长格式的命令选项名称，并将连字符 (–) 替换为下划线 (_)。示例如下：

- `region`
- `stack_name`
- `template_file`

参数值规则

TOML

- 布尔值可以是 `true` 或 `false`。例如 `confirm_changeset = true`。
- 对于字符串值，请使用引号 (`""`)。例如 `region = "us-west-2"`。
- 对于列表值，请使用引号 (`""`)，并使用空格 ( ) 分隔每个值。例如：`capabilities = "CAPABILITY_IAM CAPABILITY_NAMED_IAM"`。
- 对于包含键值对列表的值，这些键值对以空格分隔 ( )，并且每对的值用编码的引号 (`\" \"`) 括起。例如 `tags = "project=\"my-application\" stage=\"production\""`。
- 对于可以多次指定的参数值，值是参数数组。例如：`image_repositories = ["my-function-1=image-repo-1", "my-function-2=image-repo-2"]`。

YAML

- 布尔值可以是 `true` 或 `false`。例如 `confirm_changeset: true`。

- 对于包含单个字符串值的条目，引号 ("") 是可选的。例如 `region: us-west-2`。这包括包含以单个字符串形式提供的多个键值对的条目。以下是示例：

```
$ sam deploy --tags "foo=bar hello=world"
```

```
default:
  deploy:
    parameters:
      tags: foo=bar hello=world
```

- 对于包含值列表的条目或可在单个命令中多次使用的条目，请将其指定为字符串列表。

  以下是示例：

```
$ sam remote invoke --parameter "InvocationType=Event" --parameter "LogType=None"
```

```
default:
  remote_invoke:
    parameters:
        parameter:
        - InvocationType=Event
        - LogType=None
```

配置优先级

配置值时，以下优先级适用：

- 在命令行中提供的参数值优先于配置文件和模板文件 `Parameters` 部分中相应的值。
- 如果在命令行或带有 `parameter_overrides` 键的配置文件中使用 `--parameter-overrides` 选项，则其值优先于模板文件 `Parameters` 部分中的值。
- 在配置文件中，为特定命令提供的条目优先于全局条目。在以下示例中，`sam deploy` 命令使用堆栈名称 `my-app-stack`。

  TOML

```
[default.global.parameters]
stack_name = "common-stack"

[default.deploy.parameters]
```

```
stack_name = "my-app-stack"
```

YAML

```
default:
  global:
    parameters:
      stack_name: common-stack
  deploy:
    parameters:
      stack_name: my-app-stack
```

创建和修改配置文件

创建配置文件

使用 `sam init` 创建应用程序时，系统会创建默认 `samconfig.toml` 文件。您也可以手动创建配置文件。

修改配置文件

您可以手动修改配置文件。另外，在任何 Amazon SAM CLI 交互式流程，配置的值将显示在方括号 ([ ]) 中。如果您修改这些值，则 Amazon SAM CLI 将更新您的配置文件。

以下是使用 `sam deploy --guided` 命令的交互式流程的示例：

```
$ sam deploy --guided

Configuring SAM deploy
======================

    Looking for config file [samconfig.toml] :  Found
    Reading default arguments  :  Success

    Setting default arguments for 'sam deploy'
    =========================================
    Stack Name [sam-app]: ENTER
    AWS Region [us-west-2]: ENTER
    #Shows you resources changes to be deployed and require a 'Y' to initiate deploy
    Confirm changes before deploy [Y/n]: n
    #SAM needs permission to be able to create roles to connect to the resources in
 your template
```

```
    Allow SAM CLI IAM role creation [Y/n]: ENTER
    #Preserves the state of previously provisioned resources when an operation fails
    Disable rollback [y/N]: ENTER
    HelloWorldFunction may not have authorization defined, Is this okay? [y/N]: y
    Save arguments to configuration file [Y/n]: ENTER
    SAM configuration file [samconfig.toml]: ENTER
    SAM configuration environment [default]: ENTER
```

修改配置文件时， Amazon SAM CLI 按如下方式处理全局值：

- 如果参数值存在于配置文件的`global`部分中，则 Amazon SAM CLI 不会将值写入特定的命令部分。
- 如果参数值同时存在于`global`和特定的命令部分中，则 Amazon SAM CLI 删除特定条目以使用全局值。

管理 Amazon SAM CLI 版本

管理您的 Amazon Serverless Application Model 命令行界面 (Amazon SAM CLI) 通过升级、降级和卸载实现的版本。或者，您可以下载并安装 Amazon SAM CLI 每晚构建。

主题

- 正在升级 Amazon SAM CLI
- 正在卸载 Amazon SAM CLI
- 从使用中切换 Homebrew 来管理 Amazon SAM CLI
- 管理 Amazon SAM CLI 每晚构建
- 正在安装 Amazon SAM CLI 使用进入虚拟环境 pip
- 管理 Amazon SAM CLI 替换为 Homebrew
- 故障排除

正在升级 Amazon SAM CLI

Linux

要升级 Amazon SAM CLI 在 Linux 上，按照中的安装说明进行操作安装 Amazon SAM CLI，但在安装命令中添加`--update`选项，如下所示：

```
sudo ./sam-installation/install --update
```

macOS

的 Amazon SAM CLI 必须使用与安装方法相同的方法进行升级。我们建议您使用软件包安装程序来安装和升级 Amazon SAM CLI.

要升级 Amazon SAM CLI 使用软件包安装程序安装最新的软件包版本。有关说明，请参阅 安装 Amazon SAM CLI。

Windows

要升级 Amazon SAM CLI，安装 Amazon SAM CLI再次重复中的 Windows 安装步骤。

正在卸载 Amazon SAM CLI

Linux

要卸载 Amazon SAM CLI 在 Linux 上，必须通过运行以下命令来删除符号链接和安装目录：

1. 找到符号链接和安装路径。

   - 使用 which 命令查找符号链接：

     ```
     which sam
     ```

     输出显示 Amazon SAM 二进制文件所在的路径，例如：

     ```
      /usr/local/bin/sam
     ```

   - 使用 ls 命令查找符号链接指向的目录：

     ```
     ls -l /usr/local/bin/sam
     ```

     在下面的示例中，安装目录为 `/usr/local/aws-sam-cli`。

     ```
      lrwxrwxrwx 1 ec2-user ec2-user 49 Oct 22 09:49 /usr/local/bin/sam -> /usr/local/
     aws-sam-cli/current/bin/sam
     ```

2. 删除符号链接。

   ```
   sudo rm /usr/local/bin/sam
   ```

3. 删除安装目录。

```
sudo rm -rf /usr/local/aws-sam-cli
```

macOS

卸载 Amazon SAM CLI 通过与安装它相同的方法。我们建议您使用软件包安装程序来安装 Amazon SAM CLI.

如果你安装了 Amazon SAM CLI 使用软件包安装程序，按照以下步骤进行卸载。

要卸载 Amazon SAM CLI

1. 移除 Amazon SAM CLI 通过修改并运行以下内容来编程：

```
$ sudo rm -rf /path-to/aws-sam-cli
```

   a. *sudo*— 如果您的用户拥有写入权限，则在哪里 Amazon SAM CLI 程序已安装，sudo不是必需的。否则，sudo 是必需的。
   b. */path-to*— 指向安装位置的路径 Amazon SAM CLI 节目。默认位置是 `/usr/local`。
2. 移除 Amazon SAM CLI `$PATH`通过修改并运行以下内容：

```
$ sudo rm -rf /path-to-symlink-directory/sam
```

   a. *sudo*— 如果您的用户具有写入权限`$PATH`，则sudo不是必需的。否则，sudo 是必需的。
   b. *path-to-symlink-directory*— 您的`$PATH`环境变量。默认位置是 `/usr/local/bin`。
3. 验证 Amazon SAM CLI 通过运行以下命令将其卸载：

```
$ sam --version
command not found: sam
```

Windows

要卸载 Amazon SAM CLI 使用 Windows 设置，请按照以下步骤操作：

1. 从“开始”菜单中搜索“添加或删除程序”。
2. 选择名为 Amazon SAM Command Line Interface 的结果，然后选择卸载以启动卸载程序。
3. 确认您要卸载 Amazon SAM CLI.

从使用中切换 Homebrew 来管理 Amazon SAM CLI

如果你使用 Homebrew 安装和升级 Amazon SAM CLI，我们建议使用 Amazon 支持的方法。按照以下说明切换到支持的方法。

要从使用中切换 Homebrew

1. 按照中的正在卸载 Homebrew 已安装 Amazon SAM CLI说明卸载 Homebrew 托管版本。
2. 按照 安装 Amazon SAM CLI 中的说明使用支持的方法安装 Amazon SAM CLI。

管理 Amazon SAM CLI 每晚构建

你可以下载并安装 Amazon SAM CLI 每晚构建。它包含的预发行版本的 Amazon SAM CLI 可能不如生产版本稳定的代码。安装后，您可以通过与 `sam-nightly` 命令使用每夜构建版本。您可以安装和使用生产版本和夜间构建版本 Amazon SAM CLI 同时。

> **Note**
>
> 每夜构建版本不包含构建映像的预发行版本。因此，通过 --use-container 选项构建无服务器应用程序时，将使用构建映像的最新生产版本。

正在安装 Amazon SAM CLI 每晚构建

要安装 Amazon SAM CLI 每晚构建，请按照以下说明进行操作。

Linux

你可以安装每晚编译版本的 Amazon SAM CLI 在 Linux 使用软件包安装程序的 x86_64 平台。

要安装 Amazon SAM CLI 每晚构建

1. 从sam-cli-nightly中下载软件包安装程序 aws-sam-cli GitHub 存储库。
2. 按照安装步骤进行操作 Amazon SAM CLI来安装夜间编译包。

macOS

你可以安装每晚编译版本的 Amazon SAM CLI on macOS，使用夜间构建包安装程序。

要安装 Amazon SAM CLI 每晚构建

1. 从sam-cli-nightly中下载适用于您平台的软件包安装程序 aws-sam-cli GitHub 存储库。
2. 按照安装步骤进行操作 Amazon SAM CLI来安装夜间编译包。

Windows

的夜间构建版本 Amazon SAM CLI 可通过以下下载链接获得：Amazon SAM CLI 每晚构建。要在 Windows 上安装每夜构建版本，请执行与在 安装 Amazon SAM CLI 中相同的步骤，但请改用每夜构建版本下载链接。

要验证您是否已安装每夜构建版本，请运行 sam-nightly --version 命令。此命令的输出形式为 `1.X.Y.dev<YYYYMMDDHHmm>`，例如：

```
SAM CLI, version 1.20.0.dev202103151200
```

切换自 Homebrew 到软件包安装程序

如果你正在使用 Homebrew 安装和升级 Amazon SAM CLI nightly build 并想切换到使用软件包安装程序，请按照以下步骤操作。

要从中切换 Homebrew 到软件包安装程序

1. 卸载 Homebrew 已安装 Amazon SAM CLI 每晚构建。

   ```
   $ brew uninstall aws-sam-cli-nightly
   ```

2. 验证 Amazon SAM CLI 通过运行以下命令卸载夜间构建：

   ```
   $ sam-nightly --version
   zsh: command not found: sam-nightly
   ```

3. 按照上一节中的步骤安装 Amazon SAM CLI 每晚构建。

正在安装 Amazon SAM CLI 使用进入虚拟环境 pip

我们建议使用本机软件包安装程序来安装 Amazon SAM CLI。 如果你必须使用 pip，我们建议您安装 Amazon SAM CLI 进入虚拟环境。这样可确保干净的安装环境，并能在出现错误时提供隔离环境。

> **Note**
>
> 截至2023年10月24日， Amazon SAM CLI 正在停止支持 Python 3.7。 要了解更多信息，请参阅Amazon SAM CLI 停止支持 Python 3.7。

要安装 Amazon SAM CLI 进入虚拟环境

1. 从您选择的起始目录，创建虚拟环境并命名。

   Linux / macOS

   ```
   $ mkdir project
   $ cd project
   $ python3 -m venv venv
   ```

   Windows

   ```
   > mkdir project
   > cd project
   > py -3 -m venv venv
   ```

2. 激活虚拟环境

   Linux / macOS

   ```
   $ . venv/bin/activate
   ```

   提示符更改为向您显示虚拟环境处于活动状态。

   ```
   (venv) $
   ```

   Windows

   ```
   > venv\Scripts\activate
   ```

   提示符更改为向您显示虚拟环境处于活动状态。

   ```
   (venv) >
   ```

3. 安装 Amazon SAM CLI 进入您的虚拟环境。

```
(venv) $ pip install --upgrade aws-sam-cli
```

4. 验证 Amazon SAM CLI 已正确安装。

```
(venv) $ sam --version
SAM CLI, version 1.94.0
```

5. 您可以使用 `deactivate` 命令退出虚拟环境。不管何时启动新会话，都必须重新激活环境。

管理 Amazon SAM CLI 替换为 Homebrew

> **Note**
>
> 从 2023 年 9 月开始， Amazon 将不再维持 Amazon 托管的 Homebrew 的安装程序 Amazon SAM CLI (`aws/tap/aws-sam-cli`)。 要继续使用 Homebrew，你可以使用社区管理的安装程序 (`aws-sam-cli`)。从 2023 年 9 月起，任何 Homebrew 引用`aws/tap/aws-sam-cli`将重定向到的命令`aws-sam-cli`。
> 我们建议您使用我们支持的安装和升级方法。

正在安装 Amazon SAM CLI 使用 Homebrew

> **Note**
>
> 这些说明使用社区管理 Amazon SAM CLI Homebrew 安装程序。如需更多支持，请参阅 homebrew-core 存储库。

要安装 Amazon SAM CLI

1. 运行以下命令：

```
$ brew install aws-sam-cli
```

2. 验证安装：

```
$ sam --version
```

成功安装之后 Amazon SAM CLI，你应该会看到如下输出：

```
SAM CLI, version 1.94.0
```

正在升级 Amazon SAM CLI 使用 Homebrew

要升级 Amazon SAM CLI 使用 Homebrew，运行以下命令：

```
$ brew upgrade aws-sam-cli
```

正在卸载 Homebrew 已安装 Amazon SAM CLI

如果 Amazon SAM CLI 是使用安装的 Homebrew，请按照以下步骤将其卸载。

要卸载 Amazon SAM CLI

1. 运行以下命令：

   ```
   $ brew uninstall aws-sam-cli
   ```

2. 验证 Amazon SAM CLI 通过运行以下命令将其卸载：

   ```
   $ sam --version
   command not found: sam
   ```

切换到社区管理 Homebrew 安装程序

如果你使用的是 Amazon 托管 Homebrew 安装程序 (`aws/tap/aws-sam-cli`) 并希望继续使用 Homebrew，我们建议切换到社区管理 Homebrew 安装程序 (`aws-sam-cli`)。

要切换单个命令，请运行以下命令：

```
$ brew uninstall aws-sam-cli && brew untap aws/tap && brew cleanup aws/tap && brew
 update && brew install aws-sam-cli
```

按照这些说明单独运行每条命令。

切换到社区管理 Homebrew 安装程序

1. 卸载 Amazon 托管 Homebrew 的版本 Amazon SAM CLI:

```
$ brew uninstall aws-sam-cli
```

2. 验证 Amazon SAM CLI 已被卸载：

```
$ which sam
sam not found
```

3. 移除 Amazon 托管的 Amazon SAM CLI 点击：

```
$ brew untap aws/tap
```

如果您收到类似以下的错误，请添加 `--force` 选项并重试。

```
Error: Refusing to untap aws/tap because it contains the following installed
 formulae or casks:
aws-sam-cli-nightly
```

4. 移除 Amazon 托管安装程序的缓存文件：

```
$ brew cleanup aws/tap
```

5. 更新 Homebrew 以及所有公式：

```
$ brew update
```

6. 安装社区管理版本的 Amazon SAM CLI:

```
$ brew install aws-sam-cli
```

7. 验证 Amazon SAM CLI 已成功安装：

```
$ sam --version
SAM CLI, version 1.94.0
```

故障排除

如果您在安装或使用时遇到错误 Amazon SAM CLI，请参阅 Amazon SAM CLI 故障排除。

设置 Amazon 凭证

Amazon SAM 命令行界面 (CLI) 要求您设置 Amazon 凭据，以便它可以代表您调用 Amazon 服务。例如， Amazon SAM CLI 拨打亚马逊 S3 的电话然后 Amazon CloudFormation。

您可能已经设置了使用 Amazon 工具的 Amazon 凭据，例如其中一个 Amazon SDKs 或 Amazon CLI。如果您还没有，本主题将向您展示设置 Amazon 凭据的推荐方法。

要设置 Amazon 证书，您必须拥有要配置的 IAM 用户的访问密钥 ID 和私有访问密钥。有关访问密钥 IDs 和私有访问密钥的信息，请参阅 IAM 用户指南中的管理 IAM 用户的访问密钥。

接下来，确定是否已 Amazon CLI 安装。然后，按照下列部分之一中的说明操作：

使用 Amazon CLI

如果您已 Amazon CLI 安装，请使用`aws configure`命令并按照提示进行操作：

```
$ aws configure
AWS Access Key ID [None]: your_access_key_id
AWS Secret Access Key [None]: your_secret_access_key
Default region name [None]:
Default output format [None]:
```

有关 `aws configure` 命令的信息，请参阅《Amazon Command Line Interface 用户指南》中的快速配置 Amazon CLI。

不使用 Amazon CLI

如果您尚未 Amazon CLI 安装，则可以创建凭据文件或设置环境变量：

- 凭证文件-您可以在本地系统的 Amazon 凭据文件中设置凭据。此文件必须位于以下位置之一：
  - 在 Linux 或 macOS 上的 `~/.aws/credentials`
  - Windows 上的 `C:\Users\USERNAME\.aws\credentials`

  此文件应包含以下格式的行：

```
[default]
aws_access_key_id = your_access_key_id
aws_secret_access_key = your_secret_access_key
```

- 环境变量 – 您可以设置 AWS_ACCESS_KEY_ID 和 AWS_SECRET_ACCESS_KEY 环境变量。

  要在 Linux 或 macOS 上设置这些变量，请使用导出命令：

  ```
  export AWS_ACCESS_KEY_ID=your_access_key_id
  export AWS_SECRET_ACCESS_KEY=your_secret_access_key
  ```

  要在 Windows 上设置这些变量，请使用 set 命令：

  ```
  set AWS_ACCESS_KEY_ID=your_access_key_id
  set AWS_SECRET_ACCESS_KEY=your_secret_access_key
  ```

遥测中的应用 Amazon SAM CLI

在 Amazon，我们根据从与客户互动中学到的知识来开发和推出服务。我们使用客户反馈来迭代我们的产品。遥测是附加信息，可帮助我们更好地了解客户需求、诊断问题并提供特征，以改善客户体验。

Amazon SAM 命令行界面 (CLI) 收集遥测数据，例如一般使用指标、系统和环境信息以及错误。有关收集的遥测类型的详细信息，请参阅收集的信息类型。

的 Amazon SAM CLI 不收集个人信息，例如用户名或电子邮件地址。它也不会提取敏感的项目级信息。

客户控制是否开启遥测功能，并且可以随时更改设置。如果遥测功能保持开启状态， Amazon SAM CLI 无需任何额外的客户互动即可在后台发送遥测数据。

关闭会话的遥测功能

在 macOS 和 Linux 操作系统中，您可以关闭单个会话的遥测功能。要关闭当前会话的遥测功能，请运行以下命令将环境变量 SAM_CLI_TELEMETRY 设置为 false。对每个新终端或会话重复此命令。

```
export SAM_CLI_TELEMETRY=0
```

在所有会话中关闭配置文件的遥测功能

运行以下命令以在运行时关闭所有会话的遥测功能 Amazon SAM CLI 在您的操作系统上。

在 Linux 中关闭遥测功能

1. 运行：

```
echo "export SAM_CLI_TELEMETRY=0" >>~/.profile
```

2. 运行：

```
source ~/.profile
```

在 macOS 中关闭遥测功能

1. 运行：

```
echo "export SAM_CLI_TELEMETRY=0" >>~/.profile
```

2. 运行：

```
source ~/.profile
```

在 Windows 中关闭遥测功能

您可以使用以下命令，在终端窗口的生命周期内临时设置环境变量：

如果使用命令提示符：

```
set SAM_CLI_TELEMETRY=0
```

如果使用 PowerShell：

```
$env:SAM_CLI_TELEMETRY=0
```

要在命令提示符或中永久设置环境变量 PowerShell，请使用以下命令：

```
setx SAM_CLI_TELEMETRY 0
```

> **Note**
>
> 在终端关闭并重新打开后，更改才会生效。

收集的信息类型

- 使用情况信息 - 客户运行的通用命令和子命令。
- 错误和诊断信息 - 客户运行的命令的状态和持续时间，包括退出代码、内部异常名称和 Docker 连接故障。
- 系统和环境信息 — Python 版本、操作系统（Windows、Linux 或 macOS）、所处的环境 Amazon SAM CLI 运行（例如 Amazon CodeBuild， Amazon IDE 工具包或终端），以及使用情况属性的哈希值。

了解更多

的遥测数据 Amazon SAM CLI 收集遵守 Amazon 数据隐私政策。有关更多信息，请参阅下列内容：

- Amazon 服务条款
- 数据隐私常见问题解答

Amazon SAM CLI 故障排除

本节详细介绍了在使用、安装和管理 Amazon Serverless Application Model 命令行界面时如何对错误消息进行故障排除 (Amazon SAM CLI).

主题

- 故障排除
- 错误消息
- 警告消息

故障排除

有关以下内容的疑难解答指南 Amazon SAM CLI，请参阅 排查 安装错误。

错误消息

Curl 错误："curl：(6) 无法解析：..."

尝试调用 API Gateway 端点时，您遇到以下错误：

```
curl: (6) Could not resolve: endpointdomain (Domain name not found)
```

这意味着您向一个无效的域发送了请求。如果您的无服务器应用程序未能成功部署，或者您的 curl 命令中有错别字，则可能会发生这种情况。使用 Amazon CloudFormation 控制台或验证应用程序是否成功部署 Amazon CLI，并验证您的curl命令是否正确。

错误：找不到具有给定堆栈名称的确切资源信息

在包含单个 Lambda 函数资源的应用程序中运行 `sam remote invoke` 命令时，您遇到以下错误：

```
Error: Can't find exact resource information with given <stack-name>. Please provide
 full resource ARN or --stack-name to resolve the ambiguity.
```

可能的原因：您没有提供 **`--stack-name`** 选项。

如果没有提供函数 ARN 作为参数，则 `sam remote invoke` 命令要求提供 `--stack-name` 选项。

解决方案：提供 **`--stack-name`** 选项。

以下是示例：

```
$ sam remote invoke --stack-name sam-app

Invoking Lambda Function HelloWorldFunction



START RequestId: 40593abb-e1ad-4d99-87bd-ac032e364e82 Version: $LATEST
END RequestId: 40593abb-e1ad-4d99-87bd-ac032e364e82
REPORT RequestId: 40593abb-e1ad-4d99-87bd-ac032e364e82  Duration: 11.31 ms
  Billed Duration: 12 ms  Memory Size: 128 MB     Max Memory Used: 67 MB  Init
 Duration: 171.71 ms
{"statusCode":200,"body":"{\"message\":\"hello world\"}"}%
```

错误：无法从堆栈名称中找到资源信息

运行 `sam remote invoke` 命令并将 Lambda 函数 ARN 作为参数传递时，您遇到以下错误：

```
Error: Can't find resource information from stack name (<stack-name>) and resource id
 (<function-id>)
```

可能的原因：您在 **`samconfig.toml`** 文件中定义了堆栈名称值。

的 Amazon SAM CLI 首先检查您的`samconfig.toml`文件中是否有堆栈名称。如果已指定堆栈名称，则参数将作为逻辑 ID 值传递。

解决方案：改为传递函数的逻辑 ID。

您可以将函数的逻辑 ID 作为参数而不是函数的 ARN 传递。

解决方案：从配置文件中移除堆栈名称值。

您可以从配置文件中移除堆栈名称值。这可以防止 Amazon SAM CLI 通过将您的函数 ARN 作为逻辑 ID 值传递。

修改配置文件后，运行 `sam build`。

错误：无法创建托管资源：无法找到凭证

运行 sam deploy 命令时，您遇到以下错误：

```
Error: Failed to create managed resources: Unable to locate credentials
```

这意味着您尚未设置 Amazon 凭据来启用 Amazon SAM CLI 拨打 Amazon 服务电话。要解决此问题，您必须设置 Amazon 凭据。有关更多信息，请参阅 设置 Amazon 凭证。

错误： FileNotFoundError 在 Windows 中

在中运行命令时 Amazon SAM CLI 在 Windows 上，你可能会看到以下错误：

```
Error: FileNotFoundError
```

可能的原因： Amazon SAM CLI 可能会与超过 Windows 最大路径限制的文件路径进行交互。

解决方案：若要解决此问题，必须启用新的长路径行为。若要执行该操作，请参阅 Microsoft Windows 应用程序开发文档中的在 Windows 10 版本 1607 及更高版本中启用长路径。

错误：pip 的依赖项解析程序...

错误文本示例：

```
ERROR: pip's dependency resolver does not currently take into account all the packages
 that are installed. This behaviour is the source of the following dependency
 conflicts.
aws-sam-cli 1.58.0 requires aws-sam-translator==1.51.0, but you have aws-sam-translator
 1.58.0 which is incompatible.
aws-sam-cli 1.58.0 requires typing-extensions==3.10.0.0, but you have typing-extensions
 4.4.0 which is incompatible.
```

可能的原因：如果你使用 pip 要安装软件包，软件包之间的依赖关系可能会发生冲突。

`aws-sam-cli` 软件包的每个版本都依赖于 `aws-sam-translator` 软件包的版本。例如，`aws-sam-cli` v1.58.0 可能依赖于 `aws-sam-translator` v1.51.0。

如果你安装了 Amazon SAM CLI 使用 pip，然后安装另一个依赖于较新版本的软件包`aws-sam-translator`，将会出现以下情况：

- 将会安装较新版本的 `aws-sam-translator`。
- 当前版本的 `aws-sam-cli` 和较新版本的 `aws-sam-translator` 可能不兼容。
- 当你使用 Amazon SAM CLI，则会发生依赖关系解析器错误。

解决方案：

1. 使用 Amazon SAM CLI 本机软件包安装程序。
   a. 卸载 Amazon SAM CLI 使用 pip。有关说明，请参阅 正在卸载 Amazon SAM CLI。
   b. 安装 Amazon SAM CLI 使用本机软件包安装程序。有关说明，请参阅 安装 Amazon SAM CLI。
   c. 如有必要，请升级 Amazon SAM CLI 使用本机软件包安装程序。有关说明，请参阅 正在升级 Amazon SAM CLI。
2. 如果你必须使用 pip，我们建议您将 Amazon SAM CLI 安装到虚拟环境中。这样可确保干净的安装环境，并能在出现错误时提供隔离环境。有关说明，请参阅 正在安装 Amazon SAM CLI 使用进入虚拟环境 pip。

错误：没有‘remote’这样的命令

运行 `sam remote invoke` 命令时，您遇到以下错误：

```
$ sam remote invoke ...
2023-06-20 08:15:07 Command remote not available
Usage: sam [OPTIONS] COMMAND [ARGS]...
Try 'sam -h' for help.

Error: No such command 'remote'.
```

可能的原因：您的版本 Amazon SAM CLI 已经过时了。

的 Amazon SAM CLI `sam remote invoke`命令已发布 Amazon SAM CLI 版本 1.88.0。可以运行 `sam --version` 命令查看版本。

解决方案：升级你的 Amazon SAM CLI 到最新版本。

有关说明，请参阅 正在升级 Amazon SAM CLI。

错误：在本地运行 Amazon SAM 项目需要 Docker。 你安装好了吗？

运行 sam local start-api 命令时，您遇到以下错误：

```
Error: Running Amazon SAM projects locally requires Docker. Have you got it installed?
```

这意味着你没有 Docker 已正确安装。Docker 需要在本地测试您的应用程序。要解决此问题，请按照有关在开发主机上安装 Docker 的说明操作。有关更多信息，请参阅 安装 Docker。

错误：未满足安全限制

运行 sam deploy --guided 时，系统用问题 *`Function`* `may not have authorization defined, Is this okay? [y/N]` 向您发出提示。如果您用 **N**（默认选项）来回应此提示，将会出现以下错误：

```
Error: Security Constraints Not Satisfied
```

此提示告知您，您即将部署的应用程序可能在未经授权的情况下配置了可公开访问的 Amazon API Gateway API。用 **N** 来回应此提示，即表明您不同意继续部署。

要解决此问题，您具有以下选项：

- 在经授权的情况下配置应用程序。有关配置授权的更多信息，请参阅 使用您的 Amazon SAM 模板控制 API 访问权限。

- 如果您打算在未经授权的情况下拥有可公开访问的 API 端点，请重新开始部署并用 **Y** 来回答此问题，以表明您同意部署。

消息：身份验证令牌缺失

尝试调用 API Gateway 端点时，您遇到以下错误：

```
{"message":"Missing Authentication Token"}
```

这意味着您向正确的域发送了请求，但系统无法识别 URI。要解决此问题，请验证完整的 URL，然后使用正确的 URL 更新 curl 命令。

警告消息

警告：... Amazon 将不再维护 Homebrew 适用于 Amazon SAM ... 的安装程序

安装时 Amazon SAM CLI 使用 Homebrew，您会看到以下警告消息：

```
Warning: ... Amazon will no longer maintain the Homebrew installer for Amazon SAM (aws/
tap/aws-sam-cli).
    For AWS supported installations, use the first party installers ...
```

潜在原因： Amazon 不再维护 Homebrew 支持。

从 2023 年 9 月开始， Amazon 将不再维持 Homebrew 的安装程序 Amazon SAM CLI.

解决方案：使用 Amazon 支持的安装方法。

- 您可以在上找到 Amazon 支持的安装方法安装 Amazon SAM CLI。

解决方案：要继续使用 Homebrew，请使用社区管理的安装程序。

- 你可以使用社区管理 Homebrew 安装程序由您自行决定。有关说明，请参阅 管理 Amazon SAM CLI 替换为 Homebrew。

Amazon SAM 连接器参考

本节包含 Amazon Serverless Application Model (Amazon SAM) 连接器资源类型的参考信息。有关连接器的简介，请参阅使用 Amazon SAM 连接器管理资源权限。

连接器支持的源资源和目的地资源类型

AWS::Serverless::Connector 资源类型支持源资源资源和目的地资源之间一定数量的连接。在 Amazon SAM 模板中配置连接器时，请使用下表来参考支持的连接以及需要为每种源和目标资源类型定义的属性。有关在模板中配置连接器的更多信息，请参阅AWS::Serverless::Connector。

对于源资源和目标资源，如果在同一个模板中定义，则使用 Id 属性。或者，可以添加 Qualifier 以缩小您定义的资源范围。当资源不在同一个模板中时，请使用受支持属性的组合。

要申请新的连接，请在serverless-application-model Amazon GitHub存储库中提交新问题。

源类型	目的地类型	权限	源属性	目的地属性
AWS::ApiGateway::RestApi	AWS::Lambda::Function	Write	Id 或 Qualifier 、ResourceId 和 Type	Id 或 Arn 和 Type
AWS::ApiGateway::RestApi	AWS::Serverless::Function	Write	Id 或 Qualifier 、ResourceId 和 Type	Id 或 Arn 和 Type
AWS::ApiGatewayV2::Api	AWS::Lambda::Function	Write	Id 或 Qualifier 、ResourceId 和 Type	Id 或 Arn 和 Type
AWS::ApiGatewayV2::Api	AWS::Serverless::Function	Write	Id 或 Qualifier 、ResourceId 和 Type	Id 或 Arn 和 Type
AWS::AppSync::DataSource	AWS::DynamoDB::Table	Read	Id 或 RoleName 和 Type	Id 或 Arn 和 Type

源类型	目的地类型	权限	源属性	目的地属性
AWS::AppSync::DataSource	AWS::DynamoDB::Table	Write	Id 或 RoleName 和 Type	Id 或 Arn 和 Type
AWS::AppSync::DataSource	AWS::Events::EventBus	Write	Id 或 RoleName 和 Type	Id 或 Arn 和 Type
AWS::AppSync::DataSource	AWS::Lambda::Function	Write	Id 或 RoleName 和 Type	Id 或 Arn 和 Type
AWS::AppSync::DataSource	AWS::Serverless::Function	Write	Id 或 RoleName 和 Type	Id 或 Arn 和 Type
AWS::AppSync::DataSource	AWS::Serverless::SimpleTable	Read	Id 或 RoleName 和 Type	Id 或 Arn 和 Type
AWS::AppSync::DataSource	AWS::Serverless::SimpleTable	Write	Id 或 RoleName 和 Type	Id 或 Arn 和 Type
AWS::AppSync::GraphQLApi	AWS::Lambda::Function	Write	Id 或 ResourceId 和 Type	Id 或 Arn 和 Type
AWS::AppSync::GraphQLApi	AWS::Serverless::Function	Write	Id 或 ResourceId 和 Type	Id 或 Arn 和 Type
AWS::DynamoDB::Table	AWS::Lambda::Function	Read	Id 或 Arn 和 Type	Id 或 RoleName 和 Type

源类型	目的地类型	权限	源属性	目的地属性
AWS::DynamoDB::Table	AWS::Serverless::Function	Read	Id 或 Arn 和 Type	Id 或 RoleName 和 Type
AWS::Events::Rule	AWS::Events::EventBus	Write	Id 或 RoleName 和 Type	Id 或 Arn 和 Type
AWS::Events::Rule	AWS::Lambda::Function	Write	Id 或 Arn 和 Type	Id 或 Arn 和 Type
AWS::Events::Rule	AWS::Serverless::Function	Write	Id 或 Arn 和 Type	Id 或 Arn 和 Type
AWS::Events::Rule	AWS::Serverless::StateMachine	Write	Id 或 RoleName 和 Type	Id 或 Arn 和 Type
AWS::Events::Rule	AWS::SNS::Topic	Write	Id 或 Arn 和 Type	Id 或 Arn 和 Type
AWS::Events::Rule	AWS::SQS::Queue	Write	Id 或 Arn 和 Type	Id 或 Arn、QueueUrl 和 Type
AWS::Events::Rule	AWS::StepFunctions::StateMachine	Write	Id 或 RoleName 和 Type	Id 或 Arn 和 Type
AWS::Lambda::Function	AWS::DynamoDB::Table	Read, Write	Id 或 RoleName 和 Type	Id 或 Arn 和 Type

源类型	目的地类型	权限	源属性	目的地属性
AWS::Lambda::Function	AWS::Events::EventBus	Write	Id 或 RoleName 和 Type	Id 或 Arn 和 Type
AWS::Lambda::Function	AWS::Lambda::Function	Write	Id 或 RoleName 和 Type	Id 或 Arn 和 Type
AWS::Lambda::Function	AWS::Location::PlaceIndex	Read	Id 或 RoleName 和 Type	Id 或 Arn 和 Type
AWS::Lambda::Function	AWS::S3::Bucket	Read, Write	Id 或 RoleName 和 Type	Id 或 Arn 和 Type
AWS::Lambda::Function	AWS::Serverless::Function	Write	Id 或 RoleName 和 Type	Id 或 Arn 和 Type
AWS::Lambda::Function	AWS::Serverless::SimpleTable	Read, Write	Id 或 RoleName 和 Type	Id 或 Arn 和 Type
AWS::Lambda::Function	AWS::Serverless::StateMachine	Read, Write	Id 或 RoleName 和 Type	Id 或 Arn、Name 和 Type
AWS::Lambda::Function	AWS::SNS::Topic	Write	Id 或 RoleName 和 Type	Id 或 Arn 和 Type
AWS::Lambda::Function	AWS::SQS::Queue	Read, Write	Id 或 RoleName 和 Type	Id 或 Arn 和 Type

源类型	目的地类型	权限	源属性	目的地属性
AWS::Lambda::Function	AWS::StepFunctions::StateMachine	Read, Write	Id 或 RoleName 和 Type	Id 或 Arn、Name 和 Type
AWS::S3::Bucket	AWS::Lambda::Function	Write	Id 或 Arn 和 Type	Id 或 Arn 和 Type
AWS::S3::Bucket	AWS::Serverless::Function	Write	Id 或 Arn 和 Type	Id 或 Arn 和 Type
AWS::Serverless::Api	AWS::Lambda::Function	Write	Id 或 Qualifier 、ResourceId 和 Type	Id 或 Arn 和 Type
AWS::Serverless::Api	AWS::Serverless::Function	Write	Id 或 Qualifier 、ResourceId 和 Type	Id 或 Arn 和 Type
AWS::Serverless::Function	AWS::DynamoDB::Table	Read, Write	Id 或 RoleName 和 Type	Id 或 Arn 和 Type
AWS::Serverless::Function	AWS::Events::EventBus	Write	Id 或 RoleName 和 Type	Id 或 Arn 和 Type
AWS::Serverless::Function	AWS::Lambda::Function	Write	Id 或 RoleName 和 Type	Id 或 Arn 和 Type

源类型	目的地类型	权限	源属性	目的地属性
AWS::Serverless::Function	AWS::S3::Bucket	Read, Write	Id 或 RoleName 和 Type	Id 或 Arn 和 Type
AWS::Serverless::Function	AWS::Serverless::Function	Write	Id 或 RoleName 和 Type	Id 或 Arn 和 Type
AWS::Serverless::Function	AWS::Serverless::SimpleTable	Read, Write	Id 或 RoleName 和 Type	Id 或 Arn 和 Type
AWS::Serverless::Function	AWS::Serverless::StateMachine	Read, Write	Id 或 RoleName 和 Type	Id 或 Arn、Name 和 Type
AWS::Serverless::Function	AWS::SNS::Topic	Write	Id 或 RoleName 和 Type	Id 或 Arn 和 Type
AWS::Serverless::Function	AWS::SQS::Queue	Read, Write	Id 或 RoleName 和 Type	Id 或 Arn 和 Type
AWS::Serverless::Function	AWS::StepFunctions::StateMachine	Read, Write	Id 或 RoleName 和 Type	Id 或 Arn、Name 和 Type
AWS::Serverless::HttpApi	AWS::Lambda::Function	Write	Id 或 Qualifier 、ResourceId 和 Type	Id 或 Arn 和 Type

源类型	目的地类型	权限	源属性	目的地属性
AWS::Serverless::HttpApi	AWS::Serverless::Function	Write	Id 或 Qualifier 、ResourceId 和 Type	Id 或 Arn 和 Type
AWS::Serverless::SimpleTable	AWS::Lambda::Function	Read	Id 或 Arn 和 Type	Id 或 RoleName 和 Type
AWS::Serverless::SimpleTable	AWS::Serverless::Function	Read	Id 或 Arn 和 Type	Id 或 RoleName 和 Type
AWS::Serverless::StateMachine	AWS::DynamoDB::Table	Read, Write	Id 或 RoleName 和 Type	Id 或 Arn 和 Type
AWS::Serverless::StateMachine	AWS::Events::EventBus	Write	Id 或 RoleName 和 Type	Id 或 Arn 和 Type
AWS::Serverless::StateMachine	AWS::Lambda::Function	Write	Id 或 RoleName 和 Type	Id 或 Arn 和 Type
AWS::Serverless::StateMachine	AWS::S3::Bucket	Read, Write	Id 或 RoleName 和 Type	Id 或 Arn 和 Type

源类型	目的地类型	权限	源属性	目的地属性
AWS::Serverless::StateMachine	AWS::Serverless::Function	Write	Id 或 RoleName 和 Type	Id 或 Arn 和 Type
AWS::Serverless::StateMachine	AWS::Serverless::SimpleTable	Read, Write	Id 或 RoleName 和 Type	Id 或 Arn 和 Type
AWS::Serverless::StateMachine	AWS::Serverless::StateMachine	Read, Write	Id 或 RoleName 和 Type	Id 或 Arn、Name 和 Type
AWS::Serverless::StateMachine	AWS::SNS::Topic	Write	Id 或 RoleName 和 Type	Id 或 Arn 和 Type
AWS::Serverless::StateMachine	AWS::SQS::Queue	Write	Id 或 RoleName 和 Type	Id 或 Arn 和 Type
AWS::Serverless::StateMachine	AWS::StepFunctions::StateMachine	Read, Write	Id 或 RoleName 和 Type	Id 或 Arn、Name 和 Type
AWS::SNS::Topic	AWS::Lambda::Function	Write	Id 或 Arn 和 Type	Id 或 Arn 和 Type

源类型	目的地类型	权限	源属性	目的地属性
AWS::SNS::Topic	AWS::Serverless::Function	Write	Id 或 Arn 和 Type	Id 或 Arn 和 Type
AWS::SNS::Topic	AWS::SQS::Queue	Write	Id 或 Arn 和 Type	Id 或 Arn、QueueUrl 和 Type
AWS::SQS::Queue	AWS::Lambda::Function	Read, Write	Id 或 Arn 和 Type	Id 或 RoleName 和 Type
AWS::SQS::Queue	AWS::Serverless::Function	Read, Write	Id 或 Arn 和 Type	Id 或 RoleName 和 Type
AWS::StepFunctions::StateMachine	AWS::DynamoDB::Table	Read, Write	Id 或 RoleName 和 Type	Id 或 Arn 和 Type
AWS::StepFunctions::StateMachine	AWS::Events::EventBus	Write	Id 或 RoleName 和 Type	Id 或 Arn 和 Type
AWS::StepFunctions::StateMachine	AWS::Lambda::Function	Write	Id 或 RoleName 和 Type	Id 或 Arn 和 Type
AWS::StepFunctions::StateMachine	AWS::S3::Bucket	Read, Write	Id 或 RoleName 和 Type	Id 或 Arn 和 Type

源类型	目的地类型	权限	源属性	目的地属性
AWS::StepFunctions::StateMachine	AWS::Serverless::Function	Write	Id 或 RoleName 和 Type	Id 或 Arn 和 Type
AWS::StepFunctions::StateMachine	AWS::Serverless::SimpleTable	Read, Write	Id 或 RoleName 和 Type	Id 或 Arn 和 Type
AWS::StepFunctions::StateMachine	AWS::Serverless::StateMachine	Read, Write	Id 或 RoleName 和 Type	Id 或 Arn、Name 和 Type
AWS::StepFunctions::StateMachine	AWS::SNS::Topic	Write	Id 或 RoleName 和 Type	Id 或 Arn 和 Type
AWS::StepFunctions::StateMachine	AWS::SQS::Queue	Write	Id 或 RoleName 和 Type	Id 或 Arn 和 Type
AWS::StepFunctions::StateMachine	AWS::StepFunctions::StateMachine	Read, Write	Id 或 RoleName 和 Type	Id 或 Arn、Name 和 Type

连接器创建的 IAM 策略

本节记录了使用连接器 Amazon SAM 时创建的 Amazon Identity and Access Management (IAM) 策略。

AWS::DynamoDB::Table 到 AWS::Lambda::Function

策略类型

客户管理型策略附加在 AWS::Lambda::Function 角色上。

访问类别

Read

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:DescribeStream",
        "dynamodb:GetRecords",
        "dynamodb:GetShardIterator",
        "dynamodb:ListStreams"
      ],
      "Resource": [
        "%{Source.Arn}/stream/*"
      ]
    }
  ]
}
```

AWS::Events::Rule 到 AWS::SNS::Topic

策略类型

AWS::SNS::TopicPolicy 附加在 AWS::SNS::Topic 上。

访问类别

Write

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "events.amazonaws.com"
      },
```

```
      "Resource": "%{Destination.Arn}",
      "Action": "sns:Publish",
      "Condition": {
        "ArnEquals": {
          "aws:SourceArn": "%{Source.Arn}"
        }
      }
    }
  ]
}
```

AWS::Events::Rule 到 AWS::Events::EventBus

策略类型

客户管理型策略附加在 AWS::Events::Rule 角色上。

访问类别

Write

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "events:PutEvents"
      ],
      "Resource": [
        "%{Destination.Arn}"
      ]
    }
  ]
}
```

AWS::Events::Rule 到 AWS::StepFunctions::StateMachine

策略类型

客户管理型策略附加在 AWS::Events::Rule 角色上。

访问类别

Write

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "states:StartExecution"
      ],
      "Resource": [
        "%{Destination.Arn}"
      ]
    }
  ]
}
```

AWS::Events::Rule 到 AWS::Lambda::Function

策略类型

AWS::Lambda::Permission 附加在 AWS::Lambda::Function 上。

访问类别

Write

```
{
  "Action": "lambda:InvokeFunction",
  "Principal": "events.amazonaws.com",
  "SourceArn": "%{Source.Arn}"
}
```

AWS::Events::Rule 到 AWS::SQS::Queue

策略类型

AWS::SQS::QueuePolicy 附加在 AWS::SQS::Queue 上。

访问类别

Write

```
{
  "Statement": [
    {
      "Effect": "Allow",
```

```
      "Principal": {
        "Service": "events.amazonaws.com"
      },
      "Resource": "%{Destination.Arn}",
      "Action": "sqs:SendMessage",
      "Condition": {
        "ArnEquals": {
          "aws:SourceArn": "%{Source.Arn}"
        }
      }
    }
  ]
}
```

AWS::Lambda::Function 到 AWS::Lambda::Function

策略类型

客户管理型策略附加在 AWS::Lambda::Function 角色上。

访问类别

Write

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "lambda:InvokeAsync",
        "lambda:InvokeFunction"
      ],
      "Resource": [
        "%{Destination.Arn}"
      ]
    }
  ]
}
```

AWS::Lambda::Function 到 AWS::S3::Bucket

策略类型

客户管理型策略附加在 AWS::Lambda::Function 角色上。

访问类别

Read

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:GetObjectAcl",
        "s3:GetObjectLegalHold",
        "s3:GetObjectRetention",
        "s3:GetObjectTorrent",
        "s3:GetObjectVersion",
        "s3:GetObjectVersionAcl",
        "s3:GetObjectVersionForReplication",
        "s3:GetObjectVersionTorrent",
        "s3:ListBucket",
        "s3:ListBucketMultipartUploads",
        "s3:ListBucketVersions",
        "s3:ListMultipartUploadParts"
      ],
      "Resource": [
        "%{Destination.Arn}",
        "%{Destination.Arn}/*"
      ]
    }
  ]
}
```

Write

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:AbortMultipartUpload",
        "s3:DeleteObject",
        "s3:DeleteObjectVersion",
        "s3:PutObject",
        "s3:PutObjectLegalHold",
```

```
        "s3:PutObjectRetention",
        "s3:RestoreObject"
      ],
      "Resource": [
        "%{Destination.Arn}",
        "%{Destination.Arn}/*"
      ]
    }
  ]
}
```

AWS::Lambda::Function 到 AWS::DynamoDB::Table

策略类型

客户管理型策略附加在 AWS::Lambda::Function 角色上。

访问类别

Read

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:GetItem",
        "dynamodb:Query",
        "dynamodb:Scan",
        "dynamodb:BatchGetItem",
        "dynamodb:ConditionCheckItem",
        "dynamodb:PartiQLSelect"
      ],
      "Resource": [
        "%{Destination.Arn}",
        "%{Destination.Arn}/index/*"
      ]
    }
  ]
}
```

Write

```
{
```

```
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:PutItem",
        "dynamodb:UpdateItem",
        "dynamodb:DeleteItem",
        "dynamodb:BatchWriteItem",
        "dynamodb:PartiQLDelete",
        "dynamodb:PartiQLInsert",
        "dynamodb:PartiQLUpdate"
      ],
      "Resource": [
        "%{Destination.Arn}",
        "%{Destination.Arn}/index/*"
      ]
    }
  ]
}
```

AWS::Lambda::Function 到 AWS::SQS::Queue

策略类型

客户管理型策略附加在 AWS::Lambda::Function 角色上。

访问类别

Read

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "sqs:ReceiveMessage",
        "sqs:GetQueueAttributes"
      ],
      "Resource": [
        "%{Destination.Arn}"
      ]
    }
  ]
}
```

Write

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "sqs:DeleteMessage",
        "sqs:SendMessage",
        "sqs:ChangeMessageVisibility",
        "sqs:PurgeQueue"
      ],
      "Resource": [
        "%{Destination.Arn}"
      ]
    }
  ]
}
```

AWS::Lambda::Function 到 AWS::SNS::Topic

策略类型

客户管理型策略附加在 AWS::Lambda::Function 角色上。

访问类别

Write

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "sns:Publish"
      ],
      "Resource": [
        "%{Destination.Arn}"
      ]
    }
  ]
}
```

AWS::Lambda::Function 到 AWS::StepFunctions::StateMachine

策略类型

客户管理型策略附加在 AWS::Lambda::Function 角色上。

访问类别

Write

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "states:StartExecution",
        "states:StartSyncExecution"
      ],
      "Resource": [
        "%{Destination.Arn}"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "states:StopExecution"
      ],
      "Resource": [
        "arn:${AWS::Partition}:states:${AWS::Region}:${AWS::AccountId}:execution:
%{Destination.Name}:*"
      ]
    }
  ]
}
```

Read

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "states:DescribeStateMachine",
        "states:ListExecutions"
```

```
      ],
      "Resource": [
        "%{Destination.Arn}"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "states:DescribeExecution",
        "states:DescribeStateMachineForExecution",
        "states:GetExecutionHistory"
      ],
      "Resource": [
        "arn:${AWS::Partition}:states:${AWS::Region}:${AWS::AccountId}:execution:
%{Destination.Name}:*"
      ]
    }
  ]
}
```

AWS::Lambda::Function 到 AWS::Events::EventBus

策略类型

客户管理型策略附加在 AWS::Lambda::Function 角色上。

访问类别

Write

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "events:PutEvents"
      ],
      "Resource": [
        "%{Destination.Arn}"
      ]
    }
  ]
}
```

AWS::Lambda::Function 到 AWS::Location::PlaceIndex

策略类型

客户管理型策略附加在 AWS::Lambda::Function 角色上。

访问类别

Read

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "geo:DescribePlaceIndex",
        "geo:GetPlace",
        "geo:SearchPlaceIndexForPosition",
        "geo:SearchPlaceIndexForSuggestions",
        "geo:SearchPlaceIndexForText"
      ],
      "Resource": [
        "%{Destination.Arn}"
      ]
    }
  ]
}
```

AWS::ApiGatewayV2::Api 到 AWS::Lambda::Function

策略类型

AWS::Lambda::Permission 附加在 AWS::Lambda::Function 上。

访问类别

Write

```
{
  "Action": "lambda:InvokeFunction",
  "Principal": "apigateway.amazonaws.com",
  "SourceArn": "arn:${AWS::Partition}:execute-api:${AWS::Region}:${AWS::AccountId}:
%{Source.ResourceId}/%{Source.Qualifier}"
}
```

AWS::ApiGateway::RestApi 到 AWS::Lambda::Function

策略类型

AWS::Lambda::Permission 附加在 AWS::Lambda::Function 上。

访问类别

Write

```
{
  "Action": "lambda:InvokeFunction",
  "Principal": "apigateway.amazonaws.com",
  "SourceArn": "arn:${AWS::Partition}:execute-api:${AWS::Region}:${AWS::AccountId}:
%{Source.ResourceId}/%{Source.Qualifier}"
}
```

AWS::SNS::Topic 到 AWS::SQS::Queue

策略类型

AWS::SQS::QueuePolicy 附加在 AWS::SQS::Queue 上。

访问类别

Write

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "sns.amazonaws.com"
      },
      "Resource": "%{Destination.Arn}",
      "Action": "sqs:SendMessage",
      "Condition": {
        "ArnEquals": {
          "aws:SourceArn": "%{Source.Arn}"
        }
      }
    }
  ]
}
```

AWS::SNS::Topic 到 AWS::Lambda::Function

策略类型

AWS::Lambda::Permission 附加在 AWS::Lambda::Function 上。

访问类别

Write

```
{
  "Action": "lambda:InvokeFunction",
  "Principal": "sns.amazonaws.com",
  "SourceArn": "%{Source.Arn}"
}
```

AWS::SQS::Queue 到 AWS::Lambda::Function

策略类型

客户管理型策略附加在 AWS::Lambda::Function 角色上。

访问类别

Write

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "sqs:DeleteMessage"
      ],
      "Resource": [
        "%{Source.Arn}"
      ]
    }
  ]
}
```

Read

```
{
  "Statement": [
```

```
    {
      "Effect": "Allow",
      "Action": [
        "sqs:ReceiveMessage",
        "sqs:GetQueueAttributes"
      ],
      "Resource": [
        "%{Source.Arn}"
      ]
    }
  ]
}
```

AWS::S3::Bucket 到 AWS::Lambda::Function

策略类型

AWS::Lambda::Permission 附加在 AWS::Lambda::Function 上。

访问类别

Write

```
{
  "Action": "lambda:InvokeFunction",
  "Principal": "s3.amazonaws.com",
  "SourceArn": "%{Source.Arn}",
  "SourceAccount": "${AWS::AccountId}"
}
```

AWS::StepFunctions::StateMachine 到 AWS::Lambda::Function

策略类型

客户管理型策略附加在 AWS::StepFunctions::StateMachine 角色上。

访问类别

Write

```
{
  "Statement": [
    {
      "Effect": "Allow",
```

```
      "Action": [
        "lambda:InvokeAsync",
        "lambda:InvokeFunction"
      ],
      "Resource": [
        "%{Destination.Arn}"
      ]
    }
  ]
}
```

AWS::StepFunctions::StateMachine 到 AWS::SNS::Topic

策略类型

客户管理型策略附加在 AWS::StepFunctions::StateMachine 角色上。

访问类别

Write

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "sns:Publish"
      ],
      "Resource": [
        "%{Destination.Arn}"
      ]
    }
  ]
}
```

AWS::StepFunctions::StateMachine 到 AWS::SQS::Queue

策略类型

客户管理型策略附加在 AWS::StepFunctions::StateMachine 角色上。

访问类别

Write

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "sqs:SendMessage"
      ],
      "Resource": [
        "%{Destination.Arn}"
      ]
    }
  ]
}
```

AWS::StepFunctions::StateMachine 到 AWS::S3::Bucket

策略类型

客户管理型策略附加在 AWS::StepFunctions::StateMachine 角色上。

访问类别

Read

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:GetObjectAcl",
        "s3:GetObjectLegalHold",
        "s3:GetObjectRetention",
        "s3:GetObjectTorrent",
        "s3:GetObjectVersion",
        "s3:GetObjectVersionAcl",
        "s3:GetObjectVersionForReplication",
        "s3:GetObjectVersionTorrent",
        "s3:ListBucket",
        "s3:ListBucketMultipartUploads",
        "s3:ListBucketVersions",
        "s3:ListMultipartUploadParts"
      ],
```

```
      "Resource": [
        "%{Destination.Arn}",
        "%{Destination.Arn}/*"
      ]
    }
  ]
}
```

Write

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:AbortMultipartUpload",
        "s3:DeleteObject",
        "s3:DeleteObjectVersion",
        "s3:PutObject",
        "s3:PutObjectLegalHold",
        "s3:PutObjectRetention",
        "s3:RestoreObject"
      ],
      "Resource": [
        "%{Destination.Arn}",
        "%{Destination.Arn}/*"
      ]
    }
  ]
}
```

AWS::StepFunctions::StateMachine 到 AWS::DynamoDB::Table

策略类型

客户管理型策略附加在 AWS::StepFunctions::StateMachine 角色上。

访问类别

Read

```
{
  "Statement": [
    {
```

```
      "Effect": "Allow",
      "Action": [
        "dynamodb:GetItem",
        "dynamodb:Query",
        "dynamodb:Scan",
        "dynamodb:BatchGetItem",
        "dynamodb:ConditionCheckItem",
        "dynamodb:PartiQLSelect"
      ],
      "Resource": [
        "%{Destination.Arn}",
        "%{Destination.Arn}/index/*"
      ]
    }
  ]
}
```

Write

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:PutItem",
        "dynamodb:UpdateItem",
        "dynamodb:DeleteItem",
        "dynamodb:BatchWriteItem",
        "dynamodb:PartiQLDelete",
        "dynamodb:PartiQLInsert",
        "dynamodb:PartiQLUpdate"
      ],
      "Resource": [
        "%{Destination.Arn}",
        "%{Destination.Arn}/index/*"
      ]
    }
  ]
}
```

AWS::StepFunctions::StateMachine 到 AWS::StepFunctions::StateMachine

策略类型

客户管理型策略附加在 AWS::StepFunctions::StateMachine 角色上。

访问类别

Read

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "states:DescribeExecution"
      ],
      "Resource": [
        "arn:${AWS::Partition}:states:${AWS::Region}:${AWS::AccountId}:execution:
%{Destination.Name}:*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "events:DescribeRule"
      ],
      "Resource": [
        "arn:${AWS::Partition}:events:${AWS::Region}:${AWS::AccountId}:rule/
StepFunctionsGetEventsForStepFunctionsExecutionRule"
      ]
    }
  ]
}
```

Write

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "states:StartExecution"
      ],
      "Resource": [
        "%{Destination.Arn}"
      ]
```

```
    },
    {
      "Effect": "Allow",
      "Action": [
        "states:StopExecution"
      ],
      "Resource": [
        "arn:${AWS::Partition}:states:${AWS::Region}:${AWS::AccountId}:execution:
%{Destination.Name}:*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "events:PutTargets",
        "events:PutRule"
      ],
      "Resource": [
        "arn:${AWS::Partition}:events:${AWS::Region}:${AWS::AccountId}:rule/
StepFunctionsGetEventsForStepFunctionsExecutionRule"
      ]
    }
  ]
}
```

AWS::StepFunctions::StateMachine 到 AWS::Events::EventBus

策略类型

客户管理型策略附加在 AWS::StepFunctions::StateMachine 角色上。

访问类别

Write

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "events:PutEvents"
      ],
      "Resource": [
        "%{Destination.Arn}"
```

```
      ]
    }
  ]
}
```

AWS::AppSync::DataSource 到 AWS::DynamoDB::Table

策略类型

客户管理型策略附加在 AWS::AppSync::DataSource 角色上。

访问类别

Read

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:GetItem",
        "dynamodb:Query",
        "dynamodb:Scan",
        "dynamodb:BatchGetItem",
        "dynamodb:ConditionCheckItem",
        "dynamodb:PartiQLSelect"
      ],
      "Resource": [
        "%{Destination.Arn}",
        "%{Destination.Arn}/index/*"
      ]
    }
  ]
}
```

Write

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:PutItem",
```

```
        "dynamodb:UpdateItem",
        "dynamodb:DeleteItem",
        "dynamodb:BatchWriteItem",
        "dynamodb:PartiQLDelete",
        "dynamodb:PartiQLInsert",
        "dynamodb:PartiQLUpdate"
      ],
      "Resource": [
        "%{Destination.Arn}",
        "%{Destination.Arn}/index/*"
      ]
    }
  ]
}
```

AWS::AppSync::DataSource 到 AWS::Lambda::Function

策略类型

客户管理型策略附加在 AWS::AppSync::DataSource 角色上。

访问类别

Write

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "lambda:InvokeAsync",
        "lambda:InvokeFunction"
      ],
      "Resource": [
        "%{Destination.Arn}",
        "%{Destination.Arn}:*"
      ]
    }
  ]
}
```

AWS::AppSync::DataSource 到 AWS::Events::EventBus

策略类型

客户管理型策略附加在 AWS::AppSync::DataSource 角色上。

访问类别

Write

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "events:PutEvents"
      ],
      "Resource": [
        "%{Destination.Arn}"
      ]
    }
  ]
}
```

AWS::AppSync::GraphQLApi 到 AWS::Lambda::Function

策略类型

AWS::Lambda::Permission 附加在 AWS::Lambda::Function 上。

访问类别

Write

```
{
  "Action": "lambda:InvokeFunction",
  "Principal": "appsync.amazonaws.com",
  "SourceArn": "arn:${AWS::Partition}:appsync:${AWS::Region}:${AWS::AccountId}:apis/
%{Source.ResourceId}"
}
```

安装 Docker 以与 Amazon SAM CLI

Docker 是在您的计算机上运行容器的应用程序。With Docker， Amazon SAM 可以提供类似于容器的本地环境 Amazon Lambda ，用于构建、测试和调试您的无服务器应用程序。

> **Note**
>
> Docker 仅在本地测试应用程序和使用该`--use-container`选项构建部署包时才需要。

主题

- 安装 Docker
- 后续步骤

安装 Docker

按照以下说明进行安装 Docker 在您的操作系统上。

Linux

Docker 可在许多不同的操作系统上使用，包括大多数现代 Linux 发行版，例如 CentOS, Debian，以及 Ubuntu。 有关安装的信息 Docker 在你的特定操作系统上，请参阅 Docker Docs 网站上的 Get Docker。

要安装 Docker 在亚马逊 Linux 2 或亚马逊 Linux 2023

1. 更新实例上已安装的程序包和程序包缓存。

   ```
   $ sudo yum update -y
   ```

2. 安装最新的 Docker 社区版套餐。

   - 对于 Amazon Linux 2，运行以下命令：

     ```
     $ sudo amazon-linux-extras install docker
     ```

   - 对于 Amazon Linux 2023，运行以下命令：

     ```
     $ sudo yum install -y docker
     ```

3. 启动 Docker 服务。

   ```
   $ sudo service docker start
   ```

4. 将`ec2-user`添加到`docker`群组中这样你就可以跑了 Docker 命令而不使用sudo。

```
$ sudo usermod -a -G docker ec2-user
```

5. 通过退出并重新登录接受新的 `docker` 组权限。为此，请关闭当前的 SSH 终端窗口并在新终端窗口中重新连接到实例。您的新 SSH 会话应具有相应的 `docker` 组权限。
6. 验证 `ec2-user` 是否能在不使用 sudo 的情况下运行 Docker 命令。

```
$ docker ps
```

您应该看到以下输出，确认 Docker 已安装并正在运行：

```
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
```

> **Note**
>
> 在 Linux 上，要使用与主机不同的指令集架构来构建和运行 Lambda 函数，还需要执行其他配置步骤 Docker。 例如，要在`x86_64`计算机上运行`arm64`函数，可以运行以下命令来配置 Docker 守护程序：`docker run --rm --privileged multiarch/qemu-user-static --reset -p yes`。

如果你在安装时遇到问题 Docker，请参阅 排查 安装错误。或者，请参阅 Docker Docs 网站上的 Linux 安装后步骤中的故障排除部分。

macOS

> **Note**
>
> Docker 官方支持台式机，但首先是 Amazon SAM CLI 版本 1.47.0，你可以使用替代方案，只要它们使用 Docker 运行时间。

1. 安装 Docker

   的 Amazon SAM CLI 支持 Docker 在 macOS Sierra 10.12 或更高版本上运行。了解如何安装 Docker，请参阅 “安装” Docker 适用于 Mac 的台式机 Docker 文档网站。
2. 配置共享驱动器

的 Amazon SAM CLI 要求在共享云端硬盘中列出项目目录或任何父目录。这可能需要文件共享。有关更多信息，请参阅卷装载需要文件共享疑难解答主题，网址为 Docker 文档。

3. 验证安装

   晚于 Docker 已安装，请验证它是否正常工作。还要确认你可以运行 Docker 来自命令行的命令（例如，`docker ps`）。您无需安装、提取或拉出任何容器 — Amazon SAM CLI 根据需要自动执行此操作。

如果你在安装时遇到问题 Docker，有关更多故障排除提示，请参阅的 “疑难解答和诊断” 部分 Docker 文档网站。

Windows

> **Note**
>
> Amazon SAM 官方支持 Docker 桌面。但是，从 Amazon SAM CLI 版本 1.47.0，你可以使用替代方案，只要它们使用 Docker 运行时间。

1. 安装 Docker.

   Docker 桌面支持最新的 Windows 操作系统。对于 Windows 的旧版本，Docker 工具箱可用。选择你的 Windows 版本以获得正确的版本 Docker 安装步骤：

   - 要安装 Docker 对于 Windows 10，请参阅安装 Docker 适用于 Windows 的桌面 Docker 文档网站。
   - 要安装 Docker 对于早期版本的 Windows，请参阅 Docker上面的工具箱 Docker 工具箱 GitHub 存储库。

2. 配置共享驱动器。

   的 Amazon SAM CLI 要求在共享云端硬盘中列出项目目录或任何父目录。在某些情况下，您必须共享您的云端硬盘 Docker 才能正常运行。

3. 验证安装。

   晚于 Docker 已安装，请验证它是否正常工作。还要确认你可以运行 Docker 来自命令行的命令（例如，docker ps）。您无需安装、提取或拉出任何容器 — Amazon SAM CLI 根据需要自动执行此操作。

如果你在安装时遇到问题 Docker，有关更多故障排除提示，请参阅的 “疑难解答和诊断” 部分 Docker 文档网站。

后续步骤

有关如何安装 Amazon SAM CLI，请参阅 安装 Amazon SAM CLI。

的图像存储库 Amazon SAM

Amazon SAM 借助构建容器映像，简化无服务器应用程序的持续集成和持续交付 (CI/CD) 任务。Amazon SAM 提供的镜像包括许多支持的 Amazon Lambda 运行时的 Amazon SAM 命令行界面 (CLI) 和构建工具。这使得使用 Amazon SAM CLI 构建和打包无服务器应用程序变得更轻松。您可以将这些映像与 CI/CD 系统配合使用，以自动构建和部署 Amazon SAM 应用程序。有关示例，请参阅 使用 CI/CD 系统和管线进行部署。

Amazon SAM 构建容 URIs 器镜像使用该镜像中 Amazon SAMCLI包含的版本进行标记。如果您指定未标记的 URI，则使用最新版本。例如，`public.ecr.aws/sam/build-nodejs20.x` 使用最新的映像。但是，`public.ecr.aws/sam/build-nodejs20.x:1.24.1` 使用包含 Amazon SAM CLI 版本 1.24.1 的映像。

从版本的 1.33.0 开始 Amazon SAMCLI，这两个镜像`x86_64`和`arm64`容器镜像都可用于支持的运行时。有关更多信息，请参阅《Amazon Lambda 开发人员指南》中的 Lambda 运行时系统。

> **Note**
>
> 在 1.22.0 版本之前 Amazon SAMCLI， DockerHub 是从中 Amazon SAMCLI提取容器映像的默认存储库。从版本 1.22.0 开始，默认存储库改为 Amazon Elastic Container Public (Amazon ECR Public)。要从当前默认存储库以外的存储库中提取容器映像，可以使用带 --build-image 选项的 sam build 命令。本主题末尾的示例展示了如何使用 DockerHub 存储库映像构建应用程序。

镜像存储库 URIs

下表列出了可用于构建和打包无服务器应用程序的 Amazon ECR 公共版本容器映像。 URIs Amazon SAM

> **Note**
>
> 从 Amazon SAM CLI 版本 1.22.0 开始，Amazon ECR Public 取代了 DockerHub。如果您使用的是的早期版本 Amazon SAMCLI，我们建议您升级。

运行时	Amazon ECR Public
自定义运行时 (AL2023)	public.ecr。 aws/sam/build-提供。al2023
自定义运行时 (AL2)	public.ecr。 aws/sam/build-提供的.al2
自定义运行时	public.ecr。 aws/sam/build-已提供
Java 21	public.ecr。 aws/sam/build-java21
Java 17	public.ecr。 aws/sam/build-java17
Java 11	public.ecr。 aws/sam/build-java11
Java 8	public.ecr。 aws/sam/build-java8
.NET 9	public.ecr。 aws/sam/build-dotnet9
.NET 8	public.ecr。 aws/sam/build-dotnet8
.NET 7	public.ecr。 aws/sam/build-dotnet7
.NET 6	public.ecr。 aws/sam/build-dotnet6
Node.js 22	public.ecr。 aws/sam/build-nodejs22.x
Node.js 20	public.ecr。 aws/sam/build-nodejs20.x
Node.js 18	public.ecr。 aws/sam/build-nodejs18.x
Node.js 16	public.ecr。 aws/sam/build-nodejs16.x
Python 3.13	public.ecr。 aws/sam/build-py thon3.13
Python 3.12	public.ecr。 aws/sam/build-py thon 3.12

运行时	Amazon ECR Public
Python 3.11	public.ecr。 aws/sam/build-py thon 3.11
Python 3.10	public.ecr。 aws/sam/build-py thon 3.10
Python 3.9	public.ecr。 aws/sam/build-py thon 3.9
Python 3.8	public.ecr。 aws/sam/build-py thon3.8
Ruby 3.4	public.ecr。 aws/sam/build-ruby3.4
Ruby 3.3	public.ecr。 aws/sam/build-ruby3.3
Ruby 3.2	public.ecr。 aws/sam/build-ruby3.2

示例

以下两个示例命令使用镜像存储库中的容器镜像构建应用程序：

使用从 Amazon ECR 中提取的容器镜像构建Node.js 22应用程序：

```
$ sam build --use-container --build-image public.ecr.aws/sam/build-nodejs22.x
```

使用从 Amazon ECR 中提取的Python 3.13容器镜像构建函数资源：

```
$ sam build --use-container --build-image Function1=public.ecr.aws/sam/build-python3.13
```

使用以下方法逐步部署无服务器应用程序 Amazon SAM

Amazon Serverless Application Model (Amazon SAM) 内置CodeDeploy以提供渐进式 Amazon Lambda 部署。只需几行配置， Amazon SAM 即可完成以下操作：

- 部署 Lambda 函数的新版本，并自动创建指向新版本的别名。
- 逐步将客户流量转移到新版本，直到您确认它按预期方式运行。如果更新无法正常运行，则可以回滚更改。
- 定义转移流量前和转移流量后的测试函数，来验证新部署的代码是否已正确配置并且您的应用程序是否按预期方式运行。

- 如果触发 CloudWatch 警报，则自动回滚部署。

> **Note**
>
> 如果您通过 Amazon SAM 模板启用渐进部署，则会自动为您创建 CodeDeploy 资源。您可以直接通过查看 CodeDeploy 资源 Amazon Web Services Management Console。

示例

以下示例演示了如何使用逐步 CodeDeploy 将客户转移到您新部署的 Lambda 函数版本：

```
Resources:
MyLambdaFunction:
  Type: AWS::Serverless::Function
  Properties:
    Handler: index.handler
    Runtime: nodejs20.x
    CodeUri: s3://bucket/code.zip

    AutoPublishAlias: live

    DeploymentPreference:
      Type: Canary10Percent10Minutes
      Alarms:
        # A list of alarms that you want to monitor
        - !Ref AliasErrorMetricGreaterThanZeroAlarm
        - !Ref LatestVersionErrorMetricGreaterThanZeroAlarm
      Hooks:
        # Validation Lambda functions that are run before & after traffic shifting
        PreTraffic: !Ref PreTrafficLambdaFunction
        PostTraffic: !Ref PostTrafficLambdaFunction
```

对 Amazon SAM 模板的这些修订具有以下作用：

- `AutoPublishAlias`: 通过添加此属性并指定别名， Amazon SAM:
  - 根据对 Lambda 函数的 Amazon S3 URI 的更改，检测何时部署新代码。
  - 使用最新代码创建和发布该函数的更新版本。

- 使用您提供的名称创建别名 (除非已存在别名) 并指向 Lambda 函数的更新版本。函数调用应该使用别名限定词来利用这一功能。如果您不熟悉 Lambda 函数版本控制和别名，请参阅Amazon Lambda 函数版本控制和别名。

- `Deployment Preference Type`：在前面的示例中，10% 的客户流量会立即转移到您的新版本。10 分钟后，所有流量都将转移到新版本。但是，如果您的流量前或流量后测试失败，或者触发了 CloudWatch 警报，则会回 CodeDeploy 滚您的部署。您可以通过以下方式指定如何在版本之间转移流量：
  - `Canary`：流量在两次增量中转移。可从预定义金丝雀选项中进行选择。这些选项指定在第一次增量中转移到更新后的 Lambda 函数版本的流量百分比以及以分钟为单位的间隔；然后指定在第二次增量中转移剩余的流量。
  - `Linear`：流量使用相等的增量转移，在每次增量之间的分钟数相同。您可以从预定义的线性选项中进行选择，这些选项指定在每次增量中转移的流量百分比以及每次增量之间的分钟数。
  - `AllAtOnce`：所有流量均从原始 Lambda 函数一次性地转移到更新后的 Lambda 函数版本。

  下表概述了除示例中使用的选项之外可用的其他流量转移选项。

部署首选项类型
Canary10Percent30Minutes
Canary10Percent5Minutes
Canary10Percent10Minutes
Canary10Percent15Minutes
Linear10 10 10 PercentEvery Minus
Linear10 1 PercentEvery Minute
Linear10 2 分钟 PercentEvery
Linear10 3 分钟 PercentEvery
AllAtOnce

- `Alarms`：这些 CloudWatch 警报由部署引发的任何错误触发。遇到时，它们会自动回滚您的部署。例如，如果您正在部署的更新代码导致应用程序内出现错误。另一个例子是，您指定的任何 CloudWatch 指标Amazon Lambda或自定义指标是否超过了警报阈值。

- `Hooks`：这些是转移流量前和转移流量后的测试函数，它们在开始将流量转移到新版本之前以及完成将流量转移到新版本之后运行检查。
  - `PreTraffic`：在流量转移开始之前， CodeDeploy 调用流量挂钩前 Lambda 函数。此 Lambda 函数必须回调 CodeDeploy 并指示成功或失败。如果该函数失败，它将中止并向 Amazon CloudFormation报告故障。如果函数成功，则 CodeDeploy 继续进行流量转移。
  - `PostTraffic`: 流量转移完成后， CodeDeploy 调用流量挂钩后 Lambda 函数。这与预流量挂钩类似，在该挂钩中，函数必须回调 CodeDeploy 以报告成功或失败。使用转移流量后钩子可以运行集成测试或其他验证操作。

有关更多信息，请参阅 SAM 安全部署参考。

首次逐步部署 Lambda 函数

逐步部署 Lambda 函数时， CodeDeploy 需要先前部署的函数版本才能转移流量。因此，应通过两个步骤完成首次部署：

- 第 1 步：部署您的 Lambda 函数并使用 `AutoPublishAlias` 自动创建别名。
- 第 2 步：使用 `DeploymentPreference` 执行逐步部署。

分两步执行首次逐步部署，即可 CodeDeploy 使用之前的 Lambda 函数版本来转移流量。

第 1 步：部署 Lambda 函数

```
Resources:
MyLambdaFunction:
  Type: AWS::Serverless::Function
  Properties:
    Handler: index.handler
    Runtime: nodejs20.x
    CodeUri: s3://bucket/code.zip

    AutoPublishAlias: live
```

第 2 步：执行逐步部署

```
Resources:
MyLambdaFunction:
  Type: AWS::Serverless::Function
```

```
  Properties:
    Handler: index.handler
    Runtime: nodejs20.x
    CodeUri: s3://bucket/code.zip

    AutoPublishAlias: live

    DeploymentPreference:
      Type: Canary10Percent10Minutes
      Alarms:
        # A list of alarms that you want to monitor
        - !Ref AliasErrorMetricGreaterThanZeroAlarm
        - !Ref LatestVersionErrorMetricGreaterThanZeroAlarm
      Hooks:
        # Validation Lambda functions that are run before and after traffic shifting
        PreTraffic: !Ref PreTrafficLambdaFunction
        PostTraffic: !Ref PostTrafficLambdaFunction
```

了解更多

有关配置逐步部署的实践示例，请参阅完整 Amazon SAM 研讨会中的模块 5 – 金丝雀部署。

的重要参考说明 Amazon SAM

本节包含 Amazon Serverless Application Model (Amazon SAM) 的重要说明和公告。

主题

- 2023 年重要说明

2023 年重要说明

2023 年 10 月

Amazon SAM CLI 停止支持 Python 3.7

发布时间：2023 年 10 月 20 日

Python 3.7 已于 2023 年 6 月获得 end-of-life身份。的 Amazon SAM CLI 将停止支持 Python 3.7 2023 年 10 月 24 日。欲了解更多信息，请参阅以下网址的公告 aws-sam-cli GitHub 存储库。

此变更会影响以下用户：

- 如果你使用 Python 3.7 并安装 Amazon SAM CLI 通过`pip`。
- 如果您使用`aws-sam-cli`作为库并使用构建应用程序 Python 3.7.

如果你安装并管理 Amazon SAM CLI 通过另一种方法，您不会受到影响。

对于受影响的用户，我们建议您将开发环境升级到 Python 3.8 或更新的版本。

此更改不会影响对的支持 Python 3.7 Amazon Lambda 运行时环境。有关更多信息，请参阅《Amazon Lambda 开发人员指南》中的运行时弃用策略。

的无服务器应用程序示例 Amazon SAM

本节包括两个示例应用程序：一个用于处理 DynamoDB 事件，另一个用于处理 Amazon S3 事件。每个示例都会引导您完成创建应用程序的 step-by-step过程。此外，两者都包含有关配置事件源和 Amazon 资源的详细信息。两者都首先确定开始之前必须做什么，然后按照初始化、测试、打包和部署应用程序的步骤进行。

主题

- 使用处理 DynamoDB 事件 Amazon SAM
- 使用处理亚马逊 S3 事件 Amazon SAM

使用处理 DynamoDB 事件 Amazon SAM

使用此示例应用程序，您可以在概述和快速入门指南中的所学内容基础上进行构建，然后安装另一个示例应用程序。此应用程序由一个 Lambda 函数组成，该函数由 DynamoDB 表事件源调用。Lambda 函数非常简单，它记录通过事件源消息传入的数据。

本练习向您展示如何模拟在调用 Lambda 函数时传递给它们的事件源消息。

开始前的准备工作

请确保您已完成 安装 Amazon SAM CLI 中的所需设置。

第 1 步：初始化应用程序

在本节中，您将下载由 Amazon SAM 模板和应用程序代码组成的应用程序包。

初始化应用程序

1. 在 a 处运行以下命令 Amazon SAM CLI 命令提示符。

```
sam init \
--location gh:aws-samples/cookiecutter-aws-sam-dynamodb-python \
--no-input
```

请注意，`gh:`在上面的命令中扩展为 GitHub 网址`https://github.com/`。

2. 查看命令创建的目录的内容 (`dynamodb_event_reader/`)：

   - `template.yaml`— 定义读取 DynamoDB 应用程序 Amazon 所需的两个资源：一个 Lambda 函数和一个 DynamoDB 表。模板还定义了两个资源之间的映射。
   - `read_dynamodb_event/` 目录 – 包含 DynamoDB 应用程序代码。

第 2 步：在本地测试应用程序

要进行本地测试，请使用 Amazon SAM CLI 要生成示例 DynamoDB 事件并调用 Lambda 函数，请执行以下操作：

```
sam local generate-event dynamodb update | sam local invoke --event - ReadDynamoDBEvent
```

该`generate-event`命令会创建一条测试事件源消息，类似于将所有组件部署到 Amazon 云端时创建的消息。此事件源消息通过管道传输到 Lambda 函 ReadDynamoDBEvent数。

根据 `app.py` 中的源代码，验证预期的消息是否已打印到控制台。

第 3 步：打包应用程序

在本地测试您的应用程序后，您可以使用 Amazon SAM CLI 创建部署包，用于将应用程序部署到 Amazon 云端。

要创建 Lambda 部署包

1. 在要保存打包代码的位置创建 S3 存储桶。如果要使用现有 S3 存储桶，请跳过此步骤。

   ```
   aws s3 mb s3://bucketname
   ```

2. 通过在命令提示符处运行以下 `package` CLI 命令创建部署包。

   ```
   sam package \
       --template-file template.yaml \
       --output-template-file packaged.yaml \
       --s3-bucket bucketname
   ```

   在下一步部署应用程序时，您应指定新的模板文件 `packaged.yaml`。

第 4 步：部署应用程序

现在，您已经创建了部署包，您可以使用它来将应用程序部署到 Amazon 云端。然后，测试应用程序。

将无服务器应用程序部署到云端 Amazon

- 在 Amazon SAM CLI，使用 `deploy` CLI 命令部署您在模板中定义的所有资源。

  ```
  sam deploy \
      --template-file packaged.yaml \
      --stack-name sam-app \
      --capabilities CAPABILITY_IAM \
      --region us-east-1
  ```

  在命令中，`--capabilities` 参数允许 Amazon CloudFormation 创建 IAM 角色。

  Amazon CloudFormation 创建模板中定义的 Amazon 资源。您可以在 Amazon CloudFormation 控制台中访问这些资源的名称。

在云端测试无服务器应用程序 Amazon

1. 打开 DynamoDB 控制台。
2. 将记录插入刚创建的表中。
3. 转至表格的 “指标” 选项卡，然后选择 “查看所有 CloudWatch 指标”。在 CloudWatch 控制台中，选择 Logs 即可查看日志输出。

后续步骤

Amazon SAM GitHub 存储库包含其他示例应用程序供您下载和试用。要访问此存储库，请参阅 Amazon SAM 示例应用程序。

使用处理亚马逊 S3 事件 Amazon SAM

使用此示例应用程序，您可以在前面示例所学内容的基础上进行构建，然后安装更复杂的应用程序。此应用程序由一个 Lambda 函数组成，该函数由 Amazon S3 对象上传事件源调用。本练习向您展示如何通过 Lambda 函数访问 Amazon 资源和进行 Amazon 服务调用。

此示例无服务器应用程序处理 Amazon S3 中的对象创建事件。对于上传到存储桶的每个映像，Amazon S3 都会检测对象创建的事件并调用 Lambda 函数。Lambda 函数调用 Amazon Rekognition 以检测映像中的文本。然后，它将 Amazon Rekognition 返回的结果存储在 DynamoDB 表中。

> **Note**
>
> 使用此示例应用程序，执行步骤的顺序与前面的示例略有不同。其原因是，此示例要求先创建 Amazon 资源并配置 IAM 权限，然后才能在本地测试 Lambda 函数。我们将利用 Amazon CloudFormation 来创建资源并为您配置权限。否则，您需要手动执行此操作，然后才能在本地测试 Lambda 函数。
>
> 由于此示例更为复杂，因此在执行之前，请确保熟悉前面的示例应用程序的安装。

开始前的准备工作

请确保您已完成 安装 Amazon SAM CLI 中的所需设置。

第 1 步：初始化应用程序

在本节中，您将下载由 Amazon SAM 模板和应用程序代码组成的示例应用程序。

初始化应用程序

1. 在 a 处运行以下命令 Amazon SAM CLI 命令提示符。

```
sam init \
--location https://github.com/aws-samples/cookiecutter-aws-sam-s3-rekognition-
dynamodb-python \
--no-input
```

2. 查看命令创建的目录的内容 (`aws_sam_ocr/`)：

   - `template.yaml`— 定义 Amazon S3 应用程序所需的三种 Amazon 资源：一个 Lambda 函数、一个 Amazon S3 存储桶和一个 DynamoDB 表。模板还定义了这些资源之间的映射和权限。
   - `src/` 目录 – 包含 Amazon S3 应用程序代码。
   - `SampleEvent.json` – 示例事件源，用于本地测试。

第 2 步：打包应用程序

在本地测试此应用程序之前，必须使用 Amazon SAM CLI 创建部署包，用于将应用程序部署到 Amazon 云端。此部署创建了在本地测试应用程序所需的必要 Amazon 资源和权限。

要创建 Lambda 部署包

1. 在要保存打包代码的位置创建 S3 存储桶。如果要使用现有 S3 存储桶，请跳过此步骤。

```
aws s3 mb s3://bucketname
```

2. 通过在命令提示符处运行以下 `package` CLI 命令创建部署包。

```
sam package \
    --template-file template.yaml \
    --output-template-file packaged.yaml \
    --s3-bucket bucketname
```

   在下一步部署应用程序时，您应指定新的模板文件 `packaged.yaml`。

第 3 步：部署应用程序

现在，您已经创建了部署包，您可以使用它来将应用程序部署到 Amazon 云端。然后，您可以通过在 Amazon 云端调用应用程序来测试该应用程序。

将无服务器应用程序部署到云端 Amazon

- 在 Amazon SAM CLI，使用`deploy`命令部署您在模板中定义的所有资源。

```
sam deploy \
    --template-file packaged.yaml \
    --stack-name aws-sam-ocr \
    --capabilities CAPABILITY_IAM \
    --region us-east-1
```

  在命令中，`--capabilities`参数 Amazon CloudFormation 允许创建 IAM 角色。

  Amazon CloudFormation 创建模板中定义的 Amazon 资源。您可以在 Amazon CloudFormation 控制台中访问这些资源的名称。

在云端测试无服务器应用程序 Amazon

1. 将映像上传到您为本示例应用程序创建的 Amazon S3 存储桶。
2. 打开 DynamoDB 控制台并找到已创建的表。有关 Amazon Rekognition 返回的结果，请参阅下表。
3. 验证 DynamoDB 表中是否包含新记录，其中包含 Amazon Rekognition 在上传的映像中找到的文本。

第 4 步：在本地测试应用程序

在本地测试应用程序之前，必须先检索由创建的 Amazon 资源的名称 Amazon CloudFormation。

- 从中检索 Amazon S3 密钥名称和存储桶名称 Amazon CloudFormation。通过替换对象密钥、存储桶名称和存储桶 ARN 的值来修改 `SampleEvent.json` 文件。
- 检索 DynamoDB 表名称。此名称用于以下 `sam local invoke` 命令。

使用 Amazon SAM CLI 要生成示例 Amazon S3 事件并调用 Lambda 函数，请执行以下操作：

```
TABLE_NAME=Table name obtained from Amazon CloudFormation console sam local invoke --
event SampleEvent.json
```

`TABLE_NAME=` 部分设置 DynamoDB 表名称。`--event` 参数指定包含要传递给 Lambda 函数的测试事件消息的文件。

现在，您可以根据 Amazon Rekognition 返回的结果验证是否创建了预期的 DynamoDB 记录。

后续步骤

Amazon SAM GitHub 存储库包含其他示例应用程序供您下载和试用。要访问此存储库，请参阅 Amazon SAM 示例应用程序。

Amazon SAM CLI Terraform 支持

本节介绍在Terraform项目和Terraform云端中使用 Amazon Serverless Application Model 命令行界面 (Amazon SAMCLI)。

要提供反馈和提交功能请求，请创建 GitHub 问题。

主题

- Amazon SAM CLI 的 Terraform 支持入门
- 使用 Amazon SAM CLI 和 Terraform 进行本地调试和测试
- 将 Amazon SAM CLI 与 Serverless.tf 一起使用进行本地调试和测试
- 结合使用Amazon SAM CLI 和 Terraform 参考
- Amazon SAM CLI 对于 Terraform 支持什么？

Amazon SAM CLI 的 Terraform 支持入门

本主题介绍如何开始使用 Amazon Serverless Application Model 命令行界面 (Amazon SAMCLI) Terraform。

要提供反馈和提交功能请求，请创建 GitHub 问题。

主题

- Amazon SAM CLI Terraform 先决条件
- 将 Amazon SAM CLI 命令与 Terraform 结合使用
- 为 Terraform 项目做好准备
- 设置 Terraform Cloud

Amazon SAM CLI Terraform 先决条件

完成所有先决条件，即可开始在 Terraform 项目中使用 Amazon SAM CLI。

1. 安装或升级 Amazon SAM CLI

   要检查是否已 Amazon SAM CLI 安装，请运行以下命令：

```
$ sam --version
```

如果已安装 Amazon SAM CLI，则输出将显示一个版本。要升级到最新版本，请参阅 正在升级 Amazon SAM CLI。

有关安装 Amazon SAM CLI 及其所有先决条件的说明，请参阅 安装 Amazon SAM CLI。

2. 安装 Terraform

   要检查是否已安装 Terraform，请运行以下命令：

```
$ terraform -version
```

   要安装 Terraform，请参阅 Terraform 注册表中的安装 Terraform。

3. 安装 Docker 以进行本地测试

   Amazon SAM CLI 进行本地测试需要 Docker。要安装 Docker，请参阅 安装 Docker 以与 Amazon SAM CLI。

将 Amazon SAM CLI 命令与 Terraform 结合使用

运行支持的 Amazon SAM CLI 命令时，请使用 `--hook-name` 选项并提供 `terraform` 值。以下是示例：

```
$ sam local invoke --hook-name terraform
```

您可以按照以下步骤在 Amazon SAM CLI 配置文件中配置此选项：

```
hook_name = "terraform"
```

为 Terraform 项目做好准备

完成本主题中的步骤，即可在 Terraform 项目中使用 Amazon SAM CLI。

如果您在Terraform项目之外构建 Amazon Lambda 工件，则无需进行其他设置。使用 Amazon SAM CLI 和 Terraform 进行本地调试和测试要开始使用，请参阅 Amazon SAMCLI。

如果在 Terraform 项目中构建 Lambda 构件，您必须执行以下操作：

1. 安装 Python 3.8 或更高版本
2. 安装 Make 工具。
3. 在 Terraform 项目中定义您的 Lambda 构件构建逻辑。
4. 定义 `sam metadata` 资源以告知 Amazon SAM CLI 您的构建逻辑。
5. 使用 Amazon SAMCLI`sam build`命令来构建 Lambda 工件。

安装 Python 3.8 或更高版本

Python必须有 3.8 或更高版本才能与一起使用 Amazon SAMCLI。运行 `sam build` 时， Amazon SAM CLI 会创建包含用于构建 Lambda 构件 Python 命令的 `makefiles`。

有关安装说明，请参阅 Python 的《初学者指南》 中的下载 Python。

运行以下命令验证 Python 3.8 或更高版本是否已添加到您的计算机路径中：

```
$ python --version
```

输出应显示 Python 3.8 或更高版本。

安装 Make 工具

GNU Make 是一种控制项目可执行文件和其他非源文件生成的工具。 Amazon SAM CLI 创建 `makefiles`，其依赖此工具来构建 Lambda 构件。

如果您尚未在本地计算机上安装 Make，请在继续操作之前进行安装。

对于 Windows，您可以使用 Chocolatey 进行安装。有关说明，请参阅如何在 Windows 中安装和使用 “Make” 中的使用 Chocolatey

定义 Lambda 构件的构建逻辑

使用 `null_resource` Terraform 资源类型来定义您的 Lambda 构建逻辑。以下是使用自定义生成脚本构建 Lambda 函数的示例。

```
resource "null_resource" "build_lambda_function" {
    triggers = {
        build_number = "${timestamp()}"
    }

    provisioner "local-exec" {
```

```
        command =  substr(pathexpand("~"), 0, 1) == "/"? "./
py_build.sh \"${local.lambda_src_path}\" \"${local.building_path}\"
 \"${local.lambda_code_filename}\" Function" : "powershell.exe -File .\\PyBuild.ps1
 ${local.lambda_src_path} ${local.building_path} ${local.lambda_code_filename}
 Function"
    }
}
```

定义 sam metadata 资源

`sam metadata` 资源是一种 `null_resource` Terraform 资源类型，它为 Amazon SAM CLI 提供查找您的 Lambda 构件所需的信息。项目中的每个 Lambda 函数或层都需要一个唯一的 `sam metadata` 资源。要了解有关此资源类型的更多信息，请参阅 Terraform 注册表中的 null_resource。

要定义 sam metadata 资源

1. 以 `sam_metadata_` 开头命名您的资源，以将该资源标识为 sam metadata 资源。
2. 在资源 `triggers` 块中定义您的 Lambda 构件属性。
3. 使用 `depends_on` 参数指定包含您的 Lambda 构建逻辑的 `null_resource`。

   以下是一个示例模板：

   ```
   resource "null_resource" "sam_metadata_..." {
     triggers = {
       resource_name = resource_name
       resource_type = resource_type
       original_source_code = original_source_code
       built_output_path = built_output_path
     }
     depends_on = [
       null_resource.build_lambda_function # ref to your build logic
     ]
   }
   ```

   以下是 `sam metadata` 资源示例：

   ```
   resource "null_resource" "sam_metadata_aws_lambda_function_publish_book_review" {
       triggers = {
           resource_name = "aws_lambda_function.publish_book_review"
           resource_type = "ZIP_LAMBDA_FUNCTION"
           original_source_code = "${local.lambda_src_path}"
   ```

```
        built_output_path = "${local.building_path}/${local.lambda_code_filename}"
    }
    depends_on = [
        null_resource.build_lambda_function
    ]
}
```

您的 `sam metadata` 资源内容将因 Lambda 资源类型（函数或层）和打包类型（ZIP 或图像）而异。有关更多信息以及示例，请参阅 sam 元数据资源。

当您配置 `sam metadata` 资源并使用支持的 Amazon SAM CLI 命令时， Amazon SAM CLI 将在运行该 Amazon SAM CLI 命令之前生成元数据文件。生成此文件后，您可以使用带有未来 Amazon SAM CLI 命令的 `--skip-prepare-infra` 选项来跳过元数据生成过程并节省时间。只有在您尚未对基础设施进行任何更改（例如创建新的 Lambda 函数或新的 API 端点）时，才应使用此选项。

使用 Amazon SAM CLI 来构建 Lambda 构件

使用 Amazon SAMCLI`sam build`命令来构建 Lambda 工件。当您运行 `sam build` 时， Amazon SAM CLI 会执行以下操作：

1. 在您的 Terraform 项目中寻找 `sam metadata` 资源以了解和查找您的 Lambda 资源。
2. 启动您的 Lambda 构建逻辑以构建您的 Lambda 构件。
3. 创建用于整理Terraform项目的`.aws-sam`目录，以便与 Amazon SAMCLI`sam local`命令一起使用。

要使用 sam build 进行构建

1. 在包含 Terraform 根模块的目录中，运行以下命令：

   ```
   $ sam build --hook-name terraform
   ```

2. 要构建特定的 Lambda 函数或层，请运行以下命令

   ```
   $ sam build --hook-name terraform lambda-resource-id
   ```

   Lambda 资源 ID 可以是 Lambda 函数名称或完整 Terraform 资源地址，例如 `aws_lambda_function.list_books` 或 `module.list_book_function.aws_lambda_function.this[0]`。

如果您的函数源代码或其他 Terraform 配置文件位于 Terraform 根模块所在的目录之外，则需要指定位置。使用 `--terraform-project-root-path` 选项指定包含这些文件的顶级目录的绝对或相对路径。以下是示例：

```
$ sam build --hook-name terraform --terraform-project-root-path ~/projects/terraform/
demo
```

使用容器构建

运行 Amazon SAMCLI`sam build`命令时，您可以将配置 Amazon SAMCLI为使用本地Docker容器构建应用程序。

> **Note**
> 您一定已经安装并配置 Docker。有关说明，请参阅安装 Docker 以与 Amazon SAM CLI。

要使用容器构建

1. 创建包含 Terraform、Python、和 Make 工具的 `Dockerfile`。您还应该纳入 Lambda 函数运行时。

   以下是 `Dockerfile` 示例：

```
FROM public.ecr.aws/amazonlinux/amazonlinux:2

RUN yum -y update \
    && yum install -y unzip tar gzip bzip2-devel ed gcc gcc-c++ gcc-gfortran \
    less libcurl-devel openssl openssl-devel readline-devel xz-devel \
    zlib-devel glibc-static libcxx libcxx-devel llvm-toolset-7 zlib-static \
    && rm -rf /var/cache/yum

RUN yum -y install make \
    && yum -y install zip

RUN yum install -y yum-utils \
    && yum-config-manager --add-repo https://rpm.releases.hashicorp.com/
AmazonLinux/hashicorp.repo \
    && yum -y install terraform \
    && terraform --version

# AWS Lambda Builders
```

```
RUN amazon-linux-extras enable python3.8
RUN yum clean metadata && yum -y install python3.8
RUN curl -L get-pip.io | python3.8
RUN pip3 install aws-lambda-builders
RUN ln -s /usr/bin/python3.8 /usr/bin/python3
RUN python3 --version

VOLUME /project
WORKDIR /project

ENTRYPOINT ["sh"]
```

2. 使用 docker build 构建您的 Docker 映像。

   以下是示例：

```
$ docker build --tag terraform-build:v1 <path-to-directory-containing-Dockerfile>
```

3. 使用`--use-container`和`--build-image`选项运行 Amazon SAMCLI`sam build`命令。

   以下是示例：

```
$ sam build --use-container --build-image terraform-build:v1
```

后续步骤

要开始在 Terraform 项目中使用 Amazon SAM CLI，请参阅 使用 Amazon SAM CLI 和 Terraform 进行本地调试和测试。

设置 Terraform Cloud

我们建议您使用 Terraform v1.6.0 或更新版本。如果使用较旧版本，您必须在本地生成 Terraform 计划文件。本地计划文件为 Amazon SAM CLI提供了执行本地测试和调试所需的信息。

要生成本地计划文件

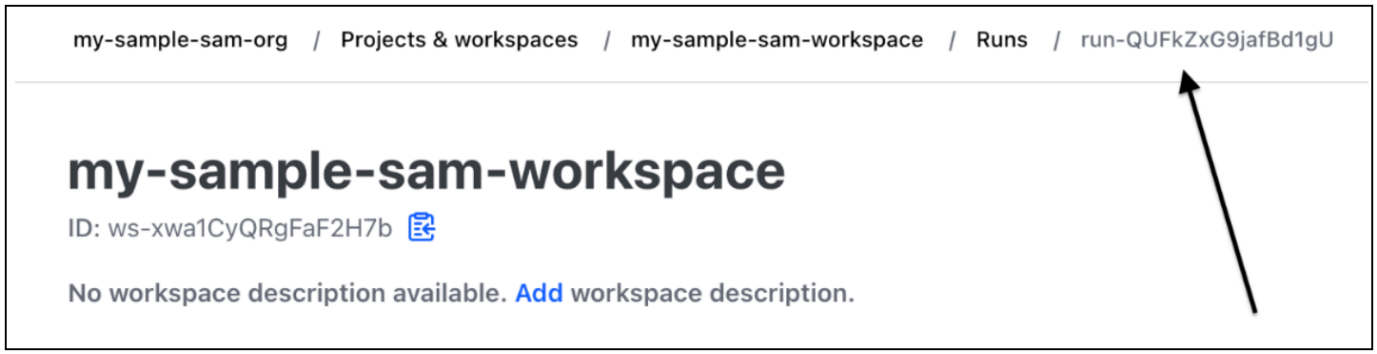
> **Note**
>
> Terraform v1.6.0 或更高版本不需要这些步骤。要开始使用 wit Amazon SAM CLI hTerraform Cloud，请参阅结合使用 Amazon SAM CLI 和 Terraform。

1. 配置 API 令牌 – 令牌的类型将取决于您的访问级别。有关更多信息，请参阅 Terraform Cloud 文档中的 API 令牌。
2. 设置 API 令牌环境变量 – 以下是命令行的示例：

```
$ export TOKEN="<api-token-value>"
```

3. 获取您的运行 ID — 在Terraform Cloud控制台中，找到您要用于的Terraform运行的运行 ID Amazon SAMCLI。

   运行 ID 位于您运行的页面导览痕迹路径中。



4. 获取计划文件 – 使用您的 API 令牌获取本地计划文件。以下是来自命令行的示例：

```
curl \
   --header "Authorization: Bearer $TOKEN" \
   --header "Content-Type: application/vnd.api+json" \
   --location \
   https://app.terraform.io/api/v2/runs/<run ID>/plan/json-output \
   > custom_plan.json
```

您现在已准备好将 Amazon SAM CLI 与 Terraform Cloud 一起使用。使用支持的 Amazon SAM CLI 命令时，请使用 `--terraform-plan-file` 选项指定本地计划文件的名称和路径。以下是示例：

```
$ sam local invoke --hook-name terraform --terraform-plan-file custom-plan.json
```

以下是使用 `sam local start-api` 命令的示例:

```
$ sam local start-api --hook-name terraform --terraform-plan-file custom-plan.json
```

有关可使用这些示例的示例应用程序，请参阅 aws-samples GitHub 存储库中的 api_gateway_v2_tf_cloud。

后续步骤

要开始使用 Amazon SAM CLI 和 Terraform Cloud，请参阅 使用 Amazon SAM CLI 和 Terraform 进行本地调试和测试。

使用 Amazon SAM CLI 和 Terraform 进行本地调试和测试

本主题介绍如何在Terraform项目中使用支持的 Amazon Serverless Application Model 命令行界面 (Amazon SAMCLI) 命令以及Terraform Cloud。

要提供反馈和提交功能请求，请创建 GitHub 问题。

主题

- 使用 sam local invoke 进行本地测试
- 使用 sam local start-api 进行本地测试
- 使用 sam local start-lambda 进行本地测试
- Terraform 限制

使用 sam local invoke 进行本地测试

> **Note**
>
> 要使用 Amazon SAM CLI 进行本地测试，必须安装并配置 Docker。有关说明，请参阅安装 Docker 以与 Amazon SAM CLI。

以下是通过传入事件进行 Lambda 函数本地测试的示例：

```
$ sam local invoke --hook-name terraform hello_world_function -e events/event.json -
```

要了解有关使用此命令的更多信息，请参阅 使用测试简介 sam local invoke。

使用 sam local start-api 进行本地测试

要将 `sam local start-api` 与 Terraform 结合使用，请运行以下命令：

```
$ sam local start-api --hook-name terraform
```

以下是示例：

```
$ sam local start-api --hook-name terraform





Running Prepare Hook to prepare the current application


Executing prepare hook of hook "terraform"


Initializing Terraform application


...
Creating terraform plan and getting JSON output


....
Generating metadata file





Unresolvable attributes discovered in project, run terraform apply to resolve them.




Finished generating metadata file. Storing in...
Prepare hook completed and metadata file generated at: ...
Mounting HelloWorldFunction at http://127.0.0.1:3000/hello [GET]
```

```
Mounting None at http://127.0.0.1:3000/hello [POST]


You can now browse to the above endpoints to invoke your functions. You do not need
 to restart/reload SAM CLI while working on your functions, changes will be reflected
 instantly/automatically. If you
used sam build before running local commands, you will need to re-run sam build for the
 changes to be picked up. You only need to restart SAM CLI if you update your AWS SAM
 template
2023-06-26 13:21:20  * Running on http://127.0.0.1:3000/ (Press CTRL+C to quit)
```

要了解有关此命令的更多信息，请参阅 使用测试简介 sam local start-api。

使用 Lambda 授权方的 Lambda 函数

对于配置为使用 Lambda 授权方的 Lambda 函数， Amazon SAM CLI 将在调用您的 Lambda 函数端点之前自动调用您的 Lambda 授权方。

- 要在中了解有关此功能的更多信息 Amazon SAMCLI，请参阅使用 Lambda 授权方的 Lambda 函数。
- 有关在 Terraform 中使用 Lambda 授权方的更多信息，请参阅 Terraform 注册表中的 Resource: aws_api_gateway_authorizer。

使用 sam local start-lambda 进行本地测试

以下是使用 Amazon Command Line Interface ()Amazon CLI在本地测试 Lambda 函数的示例：

1. 使用 Amazon SAM CLI 创建本地测试环境：

```
$ sam local start-lambda --hook-name terraform hello_world_function
```

2. 使用在本地调 Amazon CLI 用您的函数：

```
$ aws lambda invoke --function-name hello_world_function --endpoint-
url http://127.0.0.1:3001/ response.json --cli-binary-format raw-in-base64-out --
payload file://events/event.json
```

要了解有关此命令的更多信息，请参阅 使用测试简介 sam local start-lambda。

Terraform 限制

以下是将 Amazon SAM CLI 与 Terraform 一起使用时的限制：

- 链接到多个层的 Lambda 函数。
- 定义资源之间链接的 Terraform 局部变量。
- 引用尚未创建的 Lambda 函数。这包括在 REST API 资源的主体属性中定义的函数。

为避免这些限制，您可以在添加新资源时运行 `terraform apply`。

将 Amazon SAM CLI 与 Serverless.tf 一起使用进行本地调试和测试

Amazon Serverless Application Model 命令行接口 (Amazon SAMCLI) 可以与 Serverless.tf 模块一起使用，用于对 Amazon Lambda 函数和层进行本地调试和测试。支持以下 Amazon SAM CLI 命令：

- `sam build`
- `sam local invoke`
- `sam local start-api`
- `sam local start-lambda`

> **Note**
>
> Serverless.tf 版本 4.6.0 及更高版本支持 Amazon SAM CLI 集成。

要开始将 Amazon SAMCLI与 Serverless.tf 模块一起使用，请更新到最新版本 Serverless.tf 和。Amazon SAMCLI

从 serverless.tf 版本 6.0.0 开始，必须将 `create_sam_metadata` 参数设置为 `true`。这将生成 Amazon SAMCLI`sam build`命令所需的元数据资源。

要了解有关 Serverless.tf 的更多信息，请参阅 terraform-aws-lambda-module。

结合使用Amazon SAM CLI 和 Terraform 参考

本节是使用 Amazon Serverless Application Model 命令行界面 (Amazon SAMCLI) Terraform 进行本地调试和测试的参考。

要提供反馈和提交功能请求，请创建 GitHub 问题。

Amazon SAM 支持的功能参考

可以在此处找到支持与 Terraform 一起使用的 Amazon SAM CLI 功能的参考文档：

- sam build
- sam local invoke
- sam local start-api
- sam local start-lambda

Terraform 特定参考

可以在此处找到针对一起使用 Amazon SAM CLI 和 Terraform 的参考文档：

- sam 元数据资源

sam 元数据资源

本页包含 Terraform 项目所用 sam metadata resource 资源类型的参考信息。

- 有关使用 Amazon Serverless Application Model 命令行界面 (Amazon SAMCLI) 的简介Terraform，请参阅Amazon SAM CLI 对于 Terraform 支持什么？。
- 要将 Amazon SAM CLI 与 Terraform 结合使用，请参阅 使用 Amazon SAM CLI 和 Terraform 进行本地调试和测试。

主题

- 参数
- 示例

参数

参数	描述
`built_output_path`	Amazon Lambda 函数生成工件的路径。

参数	描述
docker_build_args	Docker 构建参数 JSON 对象的解码字符串。此参数是可选的。
docker_context	包含 Docker 映像构建上下文的目录的路径。
docker_file	Docker 文件的路径。路径相对于 docker_context 路径。 此参数是可选的。默认值为 Dockerfile 。
docker_tag	创建的 Docker 映像标签的值。该值为可选项。
depends_on	您的 Lambda 函数或层的构建资源的路径。要了解更多信息，请参阅 Terraform 注册表中的depends_on 参数。
original_source_code	您的 Lambda 函数的定义路径。此值可以是字符串、字符串数组或字符串形式的解码 JSON 对象。 • 对于字符串数组，由于不支持多个代码路径，因此仅使用第一个值。 • 对于 JSON 对象，还必须定义 source_code_property 。
resource_name	Lambda 函数名称。
resource_type	您的 Lambda 函数包类型的格式。接受的值为： • IMAGE_LAMBDA_FUNCTION • LAMBDA_LAYER • ZIP_LAMBDA_FUNCTION
source_code_property	JSON 对象中 Lambda 资源代码的路径。当 original_source_code 是 JSON 对象时，定义此属性。

示例

使用 ZIP 包类型引用 Lambda 函数的 sam 元数据资源

```
# Lambda function resource
resource "aws_lambda_function" "tf_lambda_func" {
```

```
  filename = "${path.module}/python/hello-world.zip"
  handler = "index.lambda_handler"
  runtime = "python3.8"
  function_name = "function_example"
  role = aws_iam_role.iam_for_lambda.arn
  depends_on = [
    null_resource.build_lambda_function # function build logic
  ]
}

# sam metadata resource
resource "null_resource" "sam_metadata_function_example" {
  triggers = {
    resource_name = "aws_lambda_function.function_example"
    resource_type = "ZIP_LAMBDA_FUNCTION"
    original_source_code = "${path.module}/python"
    built_output_path = "${path.module}/building/function_example"
  }
  depends_on = [
    null_resource.build_lambda_function # function build logic
  ]
}
```

使用映像包类型引用 Lambda 函数的 sam 元数据资源

```
resource "null_resource" "sam_metadata_function {
  triggers = {
    resource_name = "aws_lambda_function.image_function"
    resource_type = "IMAGE_LAMBDA_FUNCTION"
    docker_context = local.lambda_src_path
    docker_file = "Dockerfile"
    docker_build_args = jsonencode(var.build_args)
    docker_tag = "latest"
  }
}
```

引用 Lambda 层的 sam 元数据资源

```
resource "null_resource" "sam_metadata_layer1" {
  triggers = {
    resource_name = "aws_lambda_layer_version.layer"
    resource_type = "LAMBDA_LAYER"
    original_source_code = local.layer_src
```

```
    built_output_path = "${path.module}/${layer_build_path}"
  }
  depends_on = [null_resource.layer_build]
}
```

Amazon SAM CLI 对于 Terraform 支持什么？

使用 Amazon Serverless Application Model 命令行界面 (Amazon SAMCLI) 处理您的Terraform项目，或者Terraform Cloud对以下项目进行本地调试和测试：

- Amazon Lambda 函数和层。
- 亚马逊 API Gateway APIs。

有关 Terraform 的简介，请在 HashiCorp Terraform 网站上参阅什么是 Terraform？。

要提供反馈和提交功能请求，请创建 GitHub 问题。

> **Note**
>
> 作为集成解析步骤的一部分 Amazon SAMCLI， Amazon SAMCLI处理用户命令会生成项目文件和数据。命令输出应保持不变，但在某些环境中，环境或运行程序可能会在输出中注入其他日志或信息。

主题

- 那是什么 Amazon SAMCLI？
- 如何结合使用 Amazon SAM CLI 和 Terraform？
- 后续步骤

那是什么 Amazon SAMCLI？

Amazon SAMCLI是一种命令行工具，您可以将其与 Amazon SAM 模板和支持的第三方集成（例如Terraform用于构建和运行您的无服务器应用程序）一起使用。有关简介 Amazon SAMCLI，请参阅那是什么 Amazon SAMCLI？。

Amazon SAMCLI支持以下命令Terraform：

- `sam local invoke`— 在本地启动对 Amazon Lambda 函数资源的一次性调用。要了解有关此命令的更多信息，请参阅 使用测试简介 sam local invoke。
- `sam local start-api` – 在本地运行您的 Lambda 资源并通过本地 HTTP 服务器主机进行测试。此类测试对由 API Gateway 端点调用的 Lambda 函数很有帮助。要了解有关此命令的更多信息，请参阅 使用测试简介 sam local start-api。
- `sam local start-lambda`— 启动您的 Lambda 函数的本地终端节点，以便使用 Amazon Command Line Interface (Amazon CLI) 或在本地调用您的函数。 SDKs要了解有关此命令的更多信息，请参阅 使用测试简介 sam local start-lambda。

如何结合使用 Amazon SAM CLI 和 Terraform？

核心 Terraform 工作流程包括三个阶段：编写、计划和应用。有了对的 Amazon SAMCLI支持 Terraform，您可以利用这 Amazon SAMCLI`sam local`组命令，同时继续使用Terraform工作流程来管理应用程序 Amazon。通常，这意味着以下操作：

- 编写 – 使用 Terraform 将基础设施编写为代码。
- 测试和调试 – 使用 Amazon SAM CLI 在本地测试和调试应用程序。
- 计划 – 在应用前预览更改。
- 应用 – 配置基础设施。

有关使用 with 的示例Terraform，请参阅 C Amazon ompute 博客上的 B et HashiCorp Terraform t Amazon SAMCLI er together：和。 Amazon SAMCLI

后续步骤

要完成所有先决条件并设置 Terraform，请参阅 Amazon SAM CLI 的 Terraform 支持入门。

使用发布您的应用程序 Amazon SAM CLI

要让其他人可以查找和部署您的 Amazon SAM 应用程序，可以使用 Amazon SAM CLI 将其发布到 Amazon Serverless Application Repository。要使用发布您的应用程序 Amazon SAM CLI，则必须使用 Amazon SAM 模板对其进行定义。您还必须已在本地或 Amazon Cloud 中对应用程序进行了测试。

按照本主题中的说明创建新应用程序，创建现有应用程序的新版本，或更新现有应用程序的元数据。（您的操作取决于应用程序是否已存在于中 Amazon Serverless Application Repository，以及是否有任何应用程序元数据正在更改。）有关应用程序元数据的更多信息，请参阅 Amazon SAM 模板元数据部分属性。

先决条件

在将应用程序发布到之前， Amazon Serverless Application Repository 使用 Amazon SAM CLI，您必须具备以下条件：

- 的 Amazon SAM CLI 已安装。有关更多信息，请参阅 安装 Amazon SAM CLI。要确定是否 Amazon SAM CLI 安装完成后，运行以下命令：

  ```
  sam --version
  ```

- 有效的 Amazon SAM 模板。
- Amazon SAM 模板引用的应用程序代码和依赖关系。
- 语义版本，仅在公开共享应用程序时才需要。此值可以像 1.0 那么简单。
- 指向应用程序源代码的 URL。
- 一个 `README.md` 文件。此文件应描述客户如何使用您的应用程序，以及如何在将其部署到自己的 Amazon 账户之前对其进行配置。
- `LICENSE.txt` 文件，仅在公开共享应用程序时才需要。
- 如果您的应用程序包含任何嵌套应用程序，则这些嵌套应用程序必须已发布到 Amazon Serverless Application Repository。
- 一个有效的 Amazon Simple Storage Service (Amazon S3)存储桶策略，它为在您打包应用程序时上传到 Amazon S3 的构件授予服务读取权限。要设置此策略，请执行以下操作：

  1. 打开 Amazon S3 控制台，网址为 https://console.aws.amazon.com/s3/。
  2. 选择用于打包您的应用程序的 Amazon S3 存储桶的名称。
  3. 选择权限。

4. 在 Permissions（权限）标签页中，在 Bucket policy（存储桶策略）下，请选择 Edit（编辑）。
5. 在编辑存储桶策略页面中，将以下策略声明粘贴到策略编辑器中。在策略声明中，请确保在 `Resource` 元素中使用存储桶名称，并且在 `Condition` 元素中使用 Amazon 账户 ID。`Condition`元素中的表达式可确保仅 Amazon Serverless Application Repository 有权访问来自指定 Amazon 账户的应用程序。有关策略声明的更多信息，请参阅《IAM 用户指南》中的 IAM JSON 策略元素参考。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service":  "serverlessrepo.amazonaws.com"
            },
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::<your-bucket-name>/*",
            "Condition" : {
                "StringEquals": {
                    "aws:SourceAccount": "123456789012"
                }
            }
        }
    ]
}
```

6. 选择 Save changes（保存更改）。

发布新应用程序

第 1 步：向 Amazon SAM 模板添加分`Metadata`区

首先，在 Amazon SAM 模板中添加一个`Metadata`部分。提供要发布到 Amazon Serverless Application Repository的应用程序信息。

下面是 `Metadata` 部分的示例：

```
Metadata:
  AWS::ServerlessRepo::Application:
    Name: my-app
```

```
    Description: hello world
    Author: user1
    SpdxLicenseId: Apache-2.0
    LicenseUrl: LICENSE.txt
    ReadmeUrl: README.md
    Labels: ['tests']
    HomePageUrl: https://github.com/user1/my-app-project
    SemanticVersion: 0.0.1
    SourceCodeUrl: https://github.com/user1/my-app-project

Resources:
  HelloWorldFunction:
    Type: AWS::Lambda::Function
      Properties:
        ...
        CodeUri: source-code1
        ...
```

有关 Amazon SAM 模板`Metadata`部分的更多信息，请参阅Amazon SAM 模板元数据部分属性。

第 2 步：打包应用程序

运行以下命令 Amazon SAM CLI 命令，它将应用程序的构件上传到 Amazon S3 并输出一个名为`packaged.yaml`：

```
sam package --output-template-file packaged.yaml --s3-bucket <your-bucket-name>
```

在下一步中，要使用 `packaged.yaml` 模板文件将应用程序发布到 Amazon Serverless Application Repository。此文件与原始模板文件 (`template.yaml`) 类似，但具有一个重要区别：`CodeUri`、`LicenseUrl` 和 `ReadmeUrl` 属性指向包含相应构件的 Amazon S3 存储桶和对象。

来自示例 `packaged.yaml` 模板文件的以下代码段显示了 `CodeUri` 属性：

```
MySampleFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: s3://bucketname/fbd77a3647a4f47a352fc0ObjectGUID

...
```

第 3 步：发布应用程序

要将 Amazon SAM 应用程序的私有版本发布到 Amazon Serverless Application Repository，请运行以下命令 Amazon SAM CLI 命令：

```
sam publish --template packaged.yaml --region us-east-1
```

`sam publish` 命令的输出包括指向 Amazon Serverless Application Repository上的应用程序的链接。您也可以转到Amazon Serverless Application Repository 登录页面并搜索您的应用程序。

第 4 步：共享应用程序（可选）

默认情况下，您的应用程序设置为私有，因此其他 Amazon 账户看不到您的应用程序。要与他人共享您的应用程序，您必须将其公开或向特定的 Amazon 账户列表授予权限。

有关使用共享应用程序的信息 Amazon CLI，请参阅《Amazon Serverless Application Repository 开发人员指南》中的Amazon Serverless Application Repository 基于资源的策略示例。有关使用 Amazon Web Services Management Console共享应用程序的信息，请参阅《Amazon Serverless Application Repository 开发人员指南》中的共享应用程序。

发布现有应用程序的新版本

在将应用程序发布到之后 Amazon Serverless Application Repository，您可能需要发布该应用程序的新版本。例如，您可能更改了 Lambda 函数代码或向应用程序架构添加了新组件。

要更新您之前发布的应用程序，请使用前面详述的相同流程再次发布该应用程序。在 Amazon SAM 模板文件的 `Metadata` 部分，提供您最初发布应用程序时使用的应用程序名称，但名称要包含一个新的 `SemanticVersion` 值。

例如，假设发布的应用程序的名称为 `SampleApp`，并且 `SemanticVersion` 为 `1.0.0`。要更新该应用程序， Amazon SAM 模板必须包含应用程序名称 `SampleApp` 和 `SemanticVersion` `1.0.1`（或除 `1.0.0` 之外的任何版本）。

其他主题

- Amazon SAM 模板元数据部分属性

Amazon SAM 模板元数据部分属性

`AWS::ServerlessRepo::Application` 是元数据键，可用于指定要发布到 Amazon Serverless Application Repository的应用程序信息。

> **Note**
>
> Amazon CloudFormation `AWS::ServerlessRepo::Application`元数据密钥不支持内部函数。

属性

此表提供有关 Amazon SAM 模板`Metadata`部分属性的信息。本节是必需的，才能 Amazon Serverless Application Repository 使用以下方式向发布应用程序 Amazon SAM CLI.

属性	类型	必需	描述
`Name`	字符串	TRUE	应用程序的名称。 最小长度 = 1。最大长度 = 140。 模式：`"[a-zA-Z0-9\\-]+";`
`Description`	字符串	TRUE	关于应用程序的描述。 最小长度 = 1。最大长度 = 256。
`Author`	字符串	TRUE	发布应用程序的作者的姓名。 最小长度 = 1。最大长度 = 127。 模式：`"^[a-z0-9](([a-z0-9]|-(?!-))*[a-z0-9])?$";`
`SpdxLicenseId`	字符串	FALSE	有效的许可证标识符。要查看有效的许可证标识符列表，请参阅 Software Package Data Exchange (SPDX) 网站上的 SPDX 许可证列表。

属性	类型	必需	描述
`LicenseUrl`	字符串	FALSE	对本地许可文件的引用，或指向许可证文件的 Amazon S3 链接，其中该文件与您的应用程序的 spdxLicenseId 值相匹配。 未使用该`sam package`命令打包的 Amazon SAM 模板文件可以引用此属性的本地文件。但是，要使用 `sam publish` 命令发布应用程序，此属性必须是对 Amazon S3 存储桶的引用。 最大大小：5 MB。 您必须为此属性提供一个值，才能使您的应用程序变为公有的。请注意，在应用程序发布后，您无法更新此属性。因此，要向应用程序添加许可证，必须先将其删除，或者发布具有不同名称的新应用程序。
`ReadmeUrl`	字符串	FALSE	对本地自述文件的引用或指向自述文件的 Amazon S3 链接，其中该文件包含对应用程序及其工作原理的更详细描述。 未使用该`sam package`命令打包的 Amazon SAM 模板文件可以引用此属性的本地文件。但是，要使用 `sam publish` 命令发布，此属性必须是对 Amazon S3 存储桶的引用。 最大大小：5 MB。
`Labels`	字符串	FALSE	改善在搜索中发现应用程序的结果的标签。 最小长度 = 1。最大长度 = 127。最大标签数量：10。 模式：`"^[a-zA-Z0-9+\\-_:\\/@]+$";`
`HomePageUrl`	字符串	FALSE	包含有关应用程序的更多信息的 URL，例如应用程序 GitHub 存储库的位置。

属性	类型	必需	描述
SemanticVersion	字符串	FALSE	应用程序的语义版本。有关语义版本控制规范，请访问语义版本控制网站。 您必须为此属性提供一个值，才能使您的应用程序变为公有的。
SourceCodeUrl	字符串	FALSE	指向应用程序源代码的公共存储库的链接。

使用案例

本节列出了发布应用程序的用例，以及为该用例处理的 Metadata 属性。未针对给定用例列出的属性将被忽略。

- 创建新应用程序-如果中没有 Amazon Serverless Application Repository 与账户名称相匹配的应用程序，则会创建新的应用程序。
  - Name
  - SpdxLicenseId
  - LicenseUrl
  - Description
  - Author
  - ReadmeUrl
  - Labels
  - HomePageUrl
  - SourceCodeUrl
  - SemanticVersion
  - Amazon SAM 模板的内容（例如，任何事件源、资源和 Lambda 函数代码）

- 创建应用程序版本-如果中已经存在 Amazon Serverless Application Repository 与账户名称相匹配的应用程序，并且 SemanticVersion正在更改，则会创建应用程序版本。
  - Description
  - Author

  - `ReadmeUrl`
  - `Labels`
  - `HomePageUrl`
  - `SourceCodeUrl`
  - `SemanticVersion`
  - Amazon SAM 模板的内容（例如，任何事件源、资源和 Lambda 函数代码）

- 更新应用程序-如果中已经存在 Amazon Serverless Application Repository 与账户名称相匹配的应用程序，并且该 SemanticVersion应用程序没有更改，则会更新应用程序。
  - `Description`
  - `Author`
  - `ReadmeUrl`
  - `Labels`
  - `HomePageUrl`

示例

下面是 `Metadata` 部分的示例：

```
Metadata:
  AWS::ServerlessRepo::Application:
    Name: my-app
    Description: hello world
    Author: user1
    SpdxLicenseId: Apache-2.0
    LicenseUrl: LICENSE.txt
    ReadmeUrl: README.md
    Labels: ['tests']
    HomePageUrl: https://github.com/user1/my-app-project
    SemanticVersion: 0.0.1
    SourceCodeUrl: https://github.com/user1/my-app-project
```

的文档历史记录 Amazon SAM

下表描述此《Amazon Serverless Application Model 开发人员指南》每次发布时进行的重要更改。如需有关此文档更新的通知，您可以订阅 RSS 源。

- 上次文档更新日期：2024 年 6 月 20 日

变更	说明	日期
重组并更新了整个开发人员指南中的内容	重新组织并重构了指南，以提高可发现性和可用性。更新和改进了标题。在介绍主题和概念时提供了额外的详细信息。	2024 年 6 月 20 日
已添加 Amazon SAM CLI 支持 Ruby 3.3	Ruby 3.3 现已作为运行时和映像存储库提供。有关详细信息，请参阅 Image repositories 和 sam init。	2024 年 4 月 4 日
已添加 Amazon SAM CLI 命令选项	sam local start-api 命令有新选项可用：`--ssl-cert-file PATH`、`--ssl-key-file PATH`。此外，新的命令行选项 `--add-host LIST` 可用于 sam local invoke、sam local start-api 和 sam local start-lambda	2024 年 3 月 20 日
已添加 Amazon SAM CLI 支持.NET 8	.NET 8 现已作为运行时和映像存储库提供。不再支持 .NET Core 3.1、Node.js 14、Node.js 12、Python 3.7、Ruby 2.7 的运行时间和映像存储库。请参阅 Image repositories 和 sam init。	2024 年 2 月 22 日

已添加 Amazon SAM CLI arm64 软件包安装程序适用于 Linux	有关说明，请参阅安装 Amazon SAM CLI.	2023 年 12 月 6 日
为添加了--watch-exclude 选项 Amazon SAM CLI sam 同步命令	禁止文件和文件夹启动同步。要了解更多信息，请参阅指定不会启动同步的文件和文件夹。	2023 年 12 月 6 日
已添加--的build-in-source 选项 Amazon SAM CLI sam 同步命令	在源文件夹中构建项目以加快构建过程。要了解更多信息，请参阅通过在源文件夹中构建项目来加快构建时间。	2023 年 12 月 6 日
已添加--的build-in-source 选项 Amazon SAM CLI sam 构建命令	在源文件夹中构建项目以加快构建过程。要了解更多信息，请参阅通过在源文件夹中构建项目来加快构建时间。	2023 年 12 月 6 日
为添加了新的资源支持 Amazon SAM CLI 远程调用命令	将 `sam remote invoke` 与 Kinesis Data Streams 应用程序、Amazon SQS 队列和 Step Functions 状态机配合使用。要了解更多信息，请参阅使用 sam remote 调用。	2023 年 11 月 15 日
添加了新内容 Amazon SAM CLI 用于可共享测试事件的远程测试事件命令	使用 Amazon SAM CLI 从 EventBridge 架构注册表访问和管理可共享的测试事件，以便在中测试您的 Lambda 函数。Amazon Web Services 云要了解更多信息，请参阅使用 sam remote test-event。	2023 年 10 月 3 日
Amazon SAM CLI 支持 Terraform 现已正式上市	要了解更多相关信息 Amazon SAM CLI 支持 Terraform，请参阅 Amazon SAM CLITerraform 支持。	2023 年 9 月 5 日

已添加 Amazon SAM CLI 支持 Terraform Cloud	的 Amazon SAM CLI 现在支持本地测试 Terraform Cloud。要了解更多信息，请参阅设置 Terraform Cloud.	2023 年 9 月 5 日
已添加 YAML 文件格式支持 Amazon SAM CLI 配置文件	的 Amazon SAM CLI 现在支持 `[.yaml|.yml]` 文件格式。 正在配置 Amazon SAM CLI 和 Amazon SAM CLI 配置文件页面已更新。	2023 年 7 月 18 日
已添加 Amazon SAM CLIsam local start-api 命令支持 Terraform	什么是 Amazon SAM CLI 支持 Terraform？ 部分已更新为包括 Amazon SAM CLI `sam local start-api` 命令支持 Terraform.	2023 年 7 月 6 日
添加了新内容 Amazon SAM CLI 远程调用命令	要开始使用 `sam remote invoke`，请参阅使用 sam remote 调用。	2023 年 6 月 22 日
已添加 Amazon AppSyncGraphQL API 无服务器资源类型	创建描述如何定义的新`AWS::Serverless::GraphQLApi` 章节 GraphQL API 资源与 Amazon SAM。	2023 年 6 月 22 日
已添加 Amazon SAM CLI 支持 Ruby 3.2	更新了 sam init 页面，以纳入新的基本映像和运行时值。使用以下命令更新图片存储库页面 Ruby 3.2 亚马逊 ECR URI。	2023 年 6 月 6 日
添加了完整性验证的可选步骤 Amazon SAM CLI 软件包安装程序	更新安装 Amazon SAM CLI页面以反映可选步骤。创建验证的完整性 Amazon SAM CLI 用于记录步骤的安装程序页面。	2023 年 5 月 31 日

增加了 sam 同步选项以跳过基础设施同步	自定义每次运行时`sam sync`是否需要 Amazon CloudFormation 部署。要了解更多信息，请参阅跳过初始部 Amazon CloudFormation 署。	2023 年 3 月 23 日
增加了对 DocumentDB 事件源类型的支持	Amazon SAM 模板规范现在支持`AWS::Serverless::Function` 资源`DocumentDB` 的事件源类型。要了解更多信息，请参阅 DocumentDB。	2023 年 3 月 10 日
使用构建 Rust Lambda 函数 Cargo Lambda	使用 Amazon SAM CLI 使用来构建 Rust Lambda 函数 Cargo Lambda。 要了解更多信息，请参阅使用构建 Rust Lambda 函数 Cargo Lambda.	2023 年 2 月 23 日
在外部构建函数资源 Amazon SAM	增加了有关在使用 sam build 命令时跳过函数的指导。要了解更多信息，请参阅在外部构建函数 Amazon SAM。	2023 年 2 月 14 日
新的嵌入式连接器语法	使用新的嵌入式连接器语法来定义 `AWS::Serverless::Connector` 资源。要了解更多信息，请参阅使用 Amazon SAM 连接器管理资源权限。	2023 年 2 月 8 日
为添加了新的 sam list 命令 Amazon SAM CLI	使用 sam list 查看有关无服务器应用程序中资源的重要信息。要了解更多信息，请参阅 sam list。	2023 年 2 月 2 日

为 esbuild 添加了格式和 OutExtension 构建属性	使用 esbuild 构建 Node.js Lambda 函数现在支持 `Format` 和 `OutExtension` 构建属性。要了解更多信息，请参阅使用 esbuild 构建 Node.js Lambda 函数。	2023 年 2 月 2 日
在 Amazon SAM 模板规范中添加了运行时管理选项	为 Lambda 函数配置运行时管理选项。要了解更多信息，请参阅 `RuntimeManagementConfig` 。	2023 年 1 月 24 日
EventSource 为 AWS::Serverless::StateMachine资源添加了目标属性。	`AWS::Serverless::StateMachine` 资源类型支持 `EventBridgeRule` 和 `Schedule` 事件源的 `Target` 属性。	2023 年 1 月 13 日
为 Lambda 函数配置 SQS 轮询器的扩缩	使用 `AWS::Serverless::Function` 的 `ScalingConfig` 属性配置 SQS 轮询器的扩缩。要了解更多信息，请参阅 `ScalingConfig` 。	2023 年 1 月 12 日
使用 cfn- Amazon SAM lint 验证应用程序	你可以使用 cfn-lint 通过以下方式验证你的模板 Amazon SAM Amazon SAM CLI。 要了解更多信息，请参阅使用 cfn-lint 进行验证。	2023 年 1 月 11 日
使用 “应用洞察” 监控您的无服务器 CloudWatch 应用程序	配置 Amazon CloudWatch 应用程序见解以监控您的 Amazon SAM 应用程序。要了解更多信息，请参阅使用 Application Insights 监控您的无服务器 CloudWatch 应用程序。	2022 年 12 月 19 日

已添加 Amazon SAM CLI 适用于 macOS 的软件包安装程序	安装 Amazon SAM CLI 使用新的 macOS 软件包安装程序。要了解更多信息，请参阅安装 Amazon SAM CLI.	2022 年 12 月 6 日
增加了对 Lambda 的支持 SnapStart	配置您 SnapStart 的 Lambda 函数以创建快照，快照是初始化函数的缓存状态。要了解更多信息，请参阅 `AWS::Serverless::Function` 。	2022 年 11 月 28 日
已添加 Amazon SAM CLI 支持 nodejs18.x	Amazon SAM CLI 现在支持 nodejs18.x 运行时系统。要了解更多信息，请参阅 sam init。	2022 年 11 月 17 日
增加了有关配置访问权限和权限的指导	Amazon SAM 提供了两个选项，可简化对无服务器应用程序的访问和权限的管理。要了解更多信息，请参阅管理资源访问和权限。	2022 年 11 月 17 日
增加了对使用本机 AOT 编译功能构建 .NET 7 Lambda 函数的支持	使用原生 Ahead-of-Time (AOT) 编译来构建和打包您的.NET 7 Lambda 函数 Amazon SAM，从而缩短 Lambda 冷启动时间。要了解更多信息，请参阅使用本机 AOT 编译功能构建 .NET 7 Lambda 函数。	2022 年 11 月 15 日
已添加 Amazon SAM CLITerraform 支持本地调试和测试	使用 Amazon SAM CLI 在你的内部 Terraform 用于对您的 Lambda 函数和层执行本地调试和测试的项目。要了解更多信息，请参阅 Amazon SAM CLI Terraform 支持。	2022 年 11 月 14 日

增加了对 EventBridge 调度程序的 Amazon SAM 支持	Amazon Serverless Application Model (Amazon SAM) 模板规范提供了一种简单的简短语法，您可以使用该语法通过 S EventBridge cheduler 为 Amazon Lambda 和安排事件。 Amazon Step Functions 有关更多信息，请参阅使用计划 EventBridge 程序安排事件。	2022 年 11 月 10 日
简化了 Amazon SAM CLI 安装说明	Amazon SAM CLI 先决条件和可选步骤被移到单独的页面。支持的操作系统的安装步骤可在安装中找到 Amazon SAM CLI.	2022 年 11 月 4 日
增加了允许 Windows 10 用户使用长路径的解决方法	的 Amazon SAM CLI 应用程序模板存储库包含一些长文件路径，由`sam init`于 Windows 10 的MAX_PATH限制，这些路径在运行时可能会导致错误。有关更多信息，请参阅安装 Amazon SAM CLI	2022 年 11 月 4 日
更新了适用于首次部署的逐步部署流程	逐步部署 Lambda 函数 Amazon CodeDeploy 需要两个步骤。要了解更多信息，请参阅首次逐步部署 Lambda 函数。	2022 年 10 月 13 日
增加了适用于更多事件类型的其他 Lambda 事件筛选支持	`FilterCritera` 属性添加到 `MSK`、`MQ` 和 `SelfManagedKafka` 事件源类型。	2022 年 10 月 13 日

增加了对管道的 OpenID Connect (OIDC) 支持 Amazon SAM	Amazon SAM 支持 Bitbucket 、 GitHub Actions 以及 GitLab 持续集成和持续交付 (CI/CD) 平台的 OpenID Connect (OIDC) 用户身份验证。要了解更多信息，请参阅将 OIDC 用户帐户与管道配合 Amazon SAM 使用。	2022 年 10 月 13 日
关于 JwtConfiguration 属性的注意事项	在 `OAuth2Authorizer` 的 `JwtConfiguration` 下增加了有关定义 `issuer` 和 `audience` 属性的说明。	2022 年 10 月 7 日
函数和的新属性 StateMachine EventSource	`Enabled` 和 `State` 属性添加到 `AWS::Serverless::Function` 的 `CloudWatchEvent` 事件源。`State` 属性添加到 `AWS::Serverless::Function` 和 `AWS::Serverless::StateMachine` 的 `Schedule` 事件源。	2022 年 10 月 6 日
Amazon SAM 连接器现已正式上市	连接器是一种 Amazon SAM 抽象资源类型，标识为`AWS::Serverless::Connector` ，它提供了一种在无服务器应用程序资源之间配置权限的简单而安全的方法。要了解更多信息，请参阅使用 Amazon Serverless Application Model 连接器管理资源权限。	2022 年 10 月 6 日

向中添加了新的 sam 同步选项 Amazon SAM CLI	`--dependency-layer` 和 `--use-container` 选项添加到 sam sync。	2022 年 9 月 20 日
向中添加了新的 sam 部署选项 Amazon SAM CLI	`--on-failure` 选项添加到 sam deploy。	2022 年 9 月 9 日
esbuild 支持现已正式发布	要构建和打包 Node.js Lambda 函数，你可以使用 Amazon SAM CLI 使用 esbuild JavaScript 捆绑器。	2022 年 9 月 1 日
已更新 Amazon SAM CLI 遥测	更新了有关收集的系统和环境信息的描述，以纳入用法属性的哈希值。	2022 年 9 月 1 日
添加了对本地环境变量的支持 Amazon SAM CLI	将环境变量与 Amazon SAM CLI 在本地调用 Lambda 函数和在本地运行 API Gateway 时。	2022 年 9 月 1 日
对 Lambda 指令集架构的支持	使用 Amazon SAM CLI 为我们的指令集架构构建 Lambda 函数和 Lambda x86_64 层`arm64`。有关更多信息，请参阅`AWS::Serverless::Function` 资源类型的架构属性和`AWS::Serverless::LayerVersion` 资源类型的CompatibleArchitectures属性。	2021 年 10 月 1 日

生成示例管道配置	使用 Amazon SAM CLI 使用新的`sam pipeline bootstrap` 和`sam pipeline init`命令为多个 CI/CD 系统生成示例管道。有关更多信息，请参阅生成示例 CI/CD 管道。	2021 年 7 月 21 日
Amazon SAM CLIAmazon CDK 集成（预览版，第 2 阶段）	在公开预览版的第 2 阶段中，您现在可以使用 Amazon SAM CLI 打包和部署 Amazon CDK 应用程序。您也可以使用直接下载示例 Amazon CDK 应用程序 Amazon SAM CLI。 有关更多信息，请参阅 Amazon Cloud Development Kit (Amazon CDK)（预览）。	2021 年 7 月 13 日
支持 RabbitMQ 作为函数事件源	增加了对 RabbitMQ 作为无服务器函数事件源的支持。有关更多信息，请参阅 `AWS::Serverless::Function` 资源类型的 `MQ` 事件源的 `SourceAccessConfigurations` 属性。	2021 年 7 月 7 日
使用 Amazon ECR 构建容器映像部署无服务器应用程序	使用 Amazon ECR 构建容器映像，使用常见 CI/CD 系统（例如 Jenkins Amazon CodePipeline、CI/CD 和 Actions）部署无服务器应用程序。 GitLab GitHub 有关更多信息，请参阅部署无服务器应用程序。	2021 年 6 月 24 日

使用 Amazon 工具包调试 Amazon SAM 应用程序	Amazon Toolkits 现在支持使用集成开发环境 (IDEs) 和运行时的更多组合进行分步调试。有关更多信息，请参阅使用 Amazon 工具包。	2021 年 5 月 20 日
Amazon SAM CLIAmazon CDK 集成（预览）	你现在可以使用 Amazon SAM CLI 在本地测试和构建 Amazon CDK 应用程序。这是公开预览版。有关更多信息，请参阅Amazon Cloud Development Kit (Amazon CDK)（预览）。	2021 年 4 月 29 日
默认容器映像存储库更改为 Amazon ECR Public	默认容器镜像存储库已从更改 DockerHub 为 Amazon ECR 公用。有关更多信息，请参阅映像存储库。	2021 年 4 月 6 日
每晚 Amazon SAM CLI 构建	您现在可以安装预发行版本的 Amazon SAM CLI，它每晚建造。有关更多信息，请参阅 “安装” 下您选择的操作系统副主题的 “夜间构建” 部分 Amazon SAM CLI.	2021 年 3 月 25 日
构建容器环境变量支持	您现在可以将环境变量传递到构建容器。有关更多信息，请参阅`sam build`中的 `--container-env-var` 和 `--container-env-var-file` 选项。	2021 年 3 月 4 日

新的 Linux 安装过程	您现在可以安装 Amazon SAM CLI 使用原生 Linux 安装程序。有关更多信息，请参阅安装 Amazon SAM CLI 在 Linux 上。	2021 年 2 月 10 日
Support 支持死信队列 EventBridge	增加了对无服务器函数和状态机的死信队列 EventBridge 和`Schedule`事件源的支持。有关更多信息，请参阅 `AWS::Serverless::Function` 和 `AWS::Serverless::StateMachine` 资源类型的 `EventBridgeRule` 和 `Schedule` 事件源的 `DeadLetterConfig` 属性。	2021 年 1 月 29 日
支持自定义检查点	增加了对无服务器函数的 DynamoDB 和 Kinesis 事件源的自定义检查点的支持。有关更多信息，请参阅 `AWS::Serverless::Function` 资源类型的 `Kinesis` 和 `DynamoDB` 数据类型的 `FunctionResponseTypes` 属性。	2021 年 1 月 29 日
支持滚动窗口	增加了对无服务器函数的 DynamoDB 和 Kinesis 事件源的滚动窗口的支持。有关更多信息，请参阅 `AWS::Serverless::Function` 资源类型的 `Kinesis` 和 `DynamoDB` 数据类型的 `TumblingWindowInSeconds` 属性。	2020 年 12 月 17 日

支持热容器	使用本地测试时，增加了对温容器的支持 Amazon SAM CLI 命令`sam local start-api` 和`sam local start-lambda` 。有关更多信息，请参阅这些命令的 `--warm-containers` 选项。	2020 年 12 月 16 日
对 Lambda 容器映像的支持	添加了对 Lambda 容器映像的支持。有关更多信息，请参阅构建应用程序。	2020 年 12 月 1 日
支持代码签名	增加了对无服务器应用程序代码的代码签名和可信部署的支持。有关更多信息，请参阅为 Amazon SAM 应用程序配置代码签名。	2020 年 11 月 23 日
支持并行构建和缓存构建	通过将两个选项添加到 `sam build` 命令来提高无服务器应用程序构建的性能：`--parallel` - 此选项会并行而不是按顺序构建函数和层；以及 `--cached` - 在没有进行任何需要重建的更改时，此选项会使用先前版本中的构建构件。	2020 年 11 月 10 日

支持 Amazon MQ 和双向 TLS 身份验证	增加了对 Amazon MQ 作为无服务器函数事件源的支持。有关更多信息，请参阅 `AWS::Serverless::Function` 资源类型的 `EventSource` 和 `MQ` 数据类型。还为 API Gateway APIs 和 HTTP 添加了对相互传输层安全 (TLS) 身份验证的支持 APIs。有关更多信息，请参阅 `AWS::Serverless::Api` 资源类型的 `DomainConfiguration` 数据类型或 `AWS::Serverless::HttpApi` 资源类型的 `HttpApiDomainConfiguration` 数据类型。	2020 年 11 月 5 日
支持 HTTP 的 Lambda 授权者 APIs	增加了对 `AWS::Serverless::HttpApi` 资源类型的 Lambda 授权方的支持。有关更多信息，请参阅 Lambda 授权方示例 (`AWS::Serverless::HttpApi`)。	2020 年 10 月 27 日
支持多个配置文件和环境	增加了对多个配置文件和环境的支持，以存储其默认参数值 Amazon SAM CLI 命令。有关更多信息，请参阅 Amazon SAM CLI 配置文件。	2020 年 9 月 24 日

支持带有 Step Functions 的 X-Ray，以及控制访问权限时的参考文献 APIs	增加了对 X-Ray 作为无服务器状态机事件源的支持。有关更多信息，请参阅 `AWS::Serverless::StateMachine` 资源类型中 `Tracing` 属性。在控制访问权限时，还添加了对参考的支持 APIs。有关更多信息，请参阅 `ResourcePolicyStatement` 数据类型。	2020 年 9 月 17 日
Amazon MSK 支持	增加了对 Amazon MSK 作为无服务器函数事件源的支持。这允许 Amazon MSK 主题中的记录触发 Lambda 函数。有关更多信息，请参阅 `AWS::Serverless::Function` 资源类型的 `EventSource` 和 `MSK` 数据类型。	2020 年 8 月 13 日
Amazon EFS 支持	增加了对将 Amazon EFS 文件系统挂载到本地目录的支持。这允许 Lambda 函数代码访问和修改共享资源。有关更多信息，请参阅 `AWS::Serverless::Function` 资源类型中 `FileSystemConfigs` 属性。	2020 年 6 月 16 日
编排无服务器应用程序	增加了对通过使用 Amazon SAM创建 Step Functions 状态机来编排应用程序的支持。有关更多信息，请参阅使用 Amazon Step Functions 和 Amazon 资源类型编排资源。`AWS::Serverless::StateMachine`	2020 年 5 月 27 日

构建自定义运行时系统	增加了构建自定义运行时系统的功能。有关更多信息，请参阅构建自定义运行时系统。	2020 年 5 月 21 日
构建层	增加了构建各个 `LayerVersion` 资源的功能。有关更多信息，请参阅构建层。	2020 年 5 月 19 日
生成的 Amazon CloudFormation 资源	提供了有关 Amazon SAM 生成的 Amazon CloudFormation 资源以及如何引用这些资源的详细信息。有关更多信息，请参阅生成的 Amazon CloudFormation 资源。	2020 年 4 月 8 日
设置 Amazon 凭证	添加了设置 Amazon 凭据的说明，以防您尚未将其设置为与其他 Amazon 工具（例如 Amazon SDKs 或）一起使用 Amazon CLI。有关更多信息，请参阅设置 Amazon 凭据。	2020 年 1 月 17 日
Amazon SAM 规格和 Amazon SAM CLI 更新	从中迁移了 Amazon SAM 规范 GitHub。有关更多信息，请参阅Amazon SAM 规范。还更新了部署工作流程，对 `sam deploy` 命令进行了更改。	2019 年 11 月 25 日

用于控制 API Gateway 访问权限的新选项 APIs 和策略模板更新	添加了用于控制 API Gateway 访问权限的新选项 APIs：IAM 权限、API 密钥和资源策略。有关更多信息，请参阅控制 API Gateway 的访问权限 APIs。还更新了两个策略模板：RekognitionFacesPolicy 和 ElasticsearchHttpPostPolicy。有关更多信息，请参阅 Amazon SAM 策略模板。	2019 年 8 月 29 日
入门内容更新	更新了入门章节，改进了的安装说明 Amazon SAM CLI 还有 Hello World 教程。有关更多信息，请参阅入门 Amazon SAM。	2019 年 7 月 25 日
控制 API Gateway 的访问权限 APIs	增加了对控制 API Gateway 访问权限的支持 APIs。有关更多信息，请参阅控制 API Gateway 的访问权限 APIs。	2019 年 3 月 21 日
已`sam publish`添加到 Amazon SAM CLI	中的新`sam publish`命令 Amazon SAM CLI 简化了在中发布无服务器应用程序的 Amazon Serverless Application Repository过程。有关更多信息，请参阅使用发布无服务器应用程序 Amazon SAM CLI.	2018 年 12 月 21 日
支持嵌套应用程序和层	增加了对嵌套应用程序和层的支持。有关更多信息，请参阅使用嵌套应用程序和使用层。	2018 年 11 月 29 日

已`sam build`添加到 Amazon SAM CLI	中的新`sam build`命令 Amazon SAM CLI 简化了编译具有依赖关系的无服务器应用程序的过程，以便您可以在本地测试和部署这些应用程序。有关更多信息，请参阅构建应用程序。	2018 年 11 月 19 日
为添加了新的安装选项 Amazon SAM CLI	为 Linuxbrew (Linux)、MSI (Windows) 和 Homebrew (macOS) 添加了安装选项 Amazon SAM CLI有关更多信息，请参阅安装 Amazon SAM。 CLI.	2018 年 11 月 7 日
新指南	这是 Amazon Serverless Application Model 开发人员指南的首次发布。	2018 年 10 月 17 日

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。